AFRL-IF-RS-TR-2003-288
Final Technical Report
December 2003

AGILE OBJECTS: COMPONENT-BASED INHERENT SURVIVABILITY

University of California, San Diego

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. H548/J366

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-288 has been reviewed and is approved for publication

APPROVED: /s/
BRADLEY HARNISH
Project Engineer

FOR THE DIRECTOR: /s/
WARREN H. DEBANY, JR.
Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
DECEMBER 2003

3. REPORT TYPE AND DATES COVERED
FINAL Jun 99 – Dec 02

4. TITLE AND SUBTITLE

AGILE OBJECTS: COMPONENT-BASED INHERENT SURVIVABILITY

5. FUNDING NUMBERS
G - F30602-99-1-0534
PE - 62301E
PR - H548/J366
TA - 10
WU - 01

6. AUTHOR(S)
Andrew A. Chien
Riccardo Bettati
Jane W. S. Liu

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of California, San Diego
9500 Gilman Drive
LaJolla CA 92093-0114

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFGA
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2003-288

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Bradley Harnish/IFGA/(315) 330-1884 Bradley.Harnish@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

We have developed a framework called Agile Objects which leverages component object models and enables the construction of survivable systems that support increased application survivability through elusive technologies: location elusiveness, interface elusiveness, and dynamic elusiveness. Location elusiveness is the capability of application components to be reconfigured across distributed resources – while the application is running and preserving the performance and real-time properties of the application both across and during the migration. In short, an application can flee systems that are likely (or already) compromised, dynamically reconfiguring to continue its mission. Interface elusiveness (a.k.a. High Performance Invocation Protection) enables a component middleware system to manage automatic change and configuration of application components and distributed object interfaces to maintain application security. Dynamic elusiveness is the capability to dynamically manage the dimensions of elusiveness in response to a complex and evolving security/intrusion environment. Both location and interface elusiveness are supported by Agile Objects in dynamic form.

The project efforts have demonstrated location elusiveness, interface elusiveness, and dynamic elusiveness which enable the construction of component-based inherently survivable systems. These technologies were embedded in a component middleware which allows applications, based on component technologies, to exploit survivability capabilities.

14. SUBJECT TERMS agile objects, component middleware system, component object models, distributed object interfaces, location elusiveness, interface elusiveness, dynamic elusiveness, application survivability

15. NUMBER OF PAGES
55

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

I. Project Goals

The rapidly increasing performance of low-cost computing systems has produced a rich environment for desktop, distributed, and wide-area computing. In addition, the accelerating progress of communication technology is enabling distributed systems to be designed with a different view – one in which physical location matters little and applications move freely throughout a resource pool. In particular, inexpensive, high speed networks are becoming increasingly pervasive dramatically reducing the time for data shipping and coordination of distributed elements. Commodity commercial networks achieve 120-1200MByte/s or more (e.g. gigabit Ethernet and 10gig Ethernet) and are increasingly common in local-area networks and distributed computational infrastructures. They are also a good model for high performance network installations of today and tomorrow such as a embedded local-area networks on an AEGIS cruiser or within an aircraft.

The idea of dynamic applications roaming cyberspace begins with small programs such as "agents" and eventually encompass applications which provide useful complex network services (traditionally "server" applications). This technological vision of open resource pools have now been popularized as computational "grids". It is this vision that Agile Objects pursues – the notion that applications can be modularized at the component level and then made mobile across the network. Innovations in networking, communication, scheduling, and resource management, all embodied in Agile Objects middleware enable this vision for useful applications. We describe the goals and progress toward that vision in the Agile Objects project in the following text.

The Agile Objects project set out to and has demonstrated middleware technologies which can increase the flexibility and survivability of high performance distributed systems. The Agile Objects project has developed a range of technologies which enhance the capabilities of applications based on distributed or component object models. Specifically, Agile Objects will allow component-based applications to be ***location elusive*** (distributed without concern for performance, dynamicly redistributed in response to environmental changes, and achieve that redistribution while providing hard real-time performance guarantees), and ***interface elusive*** (change their interfaces dynamically in response to reconfiguration, attack, or change in system environment to increase their survivability in the face of physical or electronic attack). Both of these capabilities can be deployed dynamically in response to changes in the computational, network, or physical environment, providing a capability of ***dynamic elusiveness.*** These capabilities enable the construction of inherently survivable applications based on components. The component middleware enables applications to exploit location elusiveness, interface elusiveness, and dynamic elusiveness and respond flexibly to noisy information about attacks and to survive.

<u>Technical Approach</u> We employ component object frameworks for insertion of Agile object capabilities. Increasing large-scale use of component object frameworks presents an opportunity for middleware infrastructures which can automatically provide dramatically greater software system flexibility and thereby survivability. We have

developed a framework called Agile Objects which leverages component object models and enables the construction of survivable systems that support increased application survivability through elusive technologies: location elusiveness, interface elusiveness, and dynamic elusiveness. The project efforts have demonstrated location elusiveness, interface elusiveness, and dynamic elusiveness which enable the construction of component-based inherently survivable systems. These technologies were embedded in a component middleware which allows applications, based on component technologies, to exploit survivability capabilities.

Location Elusiveness is the capability of application components to be reconfigured across distributed resources -- while the application is running and preserving the performance and real-time properties of the application both across and during the migration. In short, an application can flee systems that are likely (or already) compromised, dynamically reconfiguring to continue its mission. Such capability leverages recent dramatic advances in user-level networking and open real-time systems, but also requires significant advances in component runtime systems, system resource virtualization, component migration, and dynamic management of application performance thru migration. We design, implement, and develop a component middleware system which enables online application reconfiguration to enhance application survivability.

Interface Elusiveness (later called High Performance Invocation Protection or HIPIP) enables a component middleware system to manage automatic change and configuration of application components and distributed object interfaces to maintain application security. Such automatic management is critical in an environment where the application is reconfigured in ways and into resource environments that the application designer never considered. For example, components presumed local may now be remoted, exposing formerly intra-process communication to a variety of network security attacks. The interface manipulation and binding technologies used pervasively in distributed object and component systems provide the core capability for interface elusiveness approaches, but at present there is little understanding of how to specify security properties, manage them for Agile Object systems, and use Interface Elusiveness techniques to provide application security. We have developed intellectual, analytical, and empirical frameworks to explore this technology. Prototypes which embody interface elusiveness approaches have been built and used to do empirical studies.

Dynamic Elusiveness is the capability to dynamically manage the dimensions of elusiveness in response to a complex and evolving security / intrusion environment. Both location and interface elusiveness are supported by Agile Objects in dynamic form.

II. Key Research Results

The Agile Objects project has accomplished a basic proof of concept of the key project ideas showing working systems that embody location independence and online migration, open real-time structures and pre-allocation of resources to enable rapid migration, online interface mutation for elusive interfaces, and most recently detailed analytical modeling which demonstrates that applications can be made robust under denial-of-service attacks using Agile Objects techniques. We summarize each of these research results in the following section.

I. Location Independent Application Configuration with Uniform Performance

Location Elusiveness enables applications to have no fixed location, providing no fixed target for electronic attack. Software and information assets (component objects) flow seamlessly on a fabric of information resources, dynamically reconfigured to meet the needs of the application, available physical resources, security, survivability, and resource loss. Open real-time system technology allows these capabilities to be achieved in real-time applications without loss of performance guarantees. Within the application, the movement of components is tracked through a highly decentralized information location service.

The key technologies for location elusiveness are: high performance distributed objects (efficient coupling, microsecond interaction), rapid object migration, and open systems with hard real-time guarantees. Current object brokers require 1 millisecond for an RPC; our efforts have improved this dramatically, enabling configuration flexibility for distributed applications. In addition, our Agile Object systems enable online migration of component object programs. These systems exploit a framework for object naming and migration based on highly decentralized services.

Building on our previous work in high performance user-level communication, we designed two transports for the Microsoft RPC system. These transports demonstrate the potential problems and the achievable performance for remote distributed object (or component) access in user-level networked environments. Our conclusions are that remote access can be achieved with overhead and latency not significantly higher than local. Consequently, that the basic premise of Agile Objects is feasible, that distributed applications can be implemented in a fashion to be both high performance and independent of configuration. Achieved performance levels are:

MSRPC (transport)	Null RPC time (round trip)
UDP/IP over 100Mbit Ethernet	360 microseconds
Datagram Transport over Fast Messages (user-level)	114 microseconds
Connection Oriented Transport over Fast Messages (user-level)	51 microseconds

Local RPC using LRPC	27 microseconds

These results indicate RPC overheads for our Fast Messages based implementations which are comparable to local overheads for LRPC. This means that comparable RPC rates can be achieved in both situations. In addition, the latency of the remote RPC is only 24 microseconds larger, nearly all of which can be accounted for by the physical latency of the network. Thus for decomposable distributed systems which are not tightly-coupled (most of them), remote RPC-based on user-level communication can deliver comparable performance to local RPC.

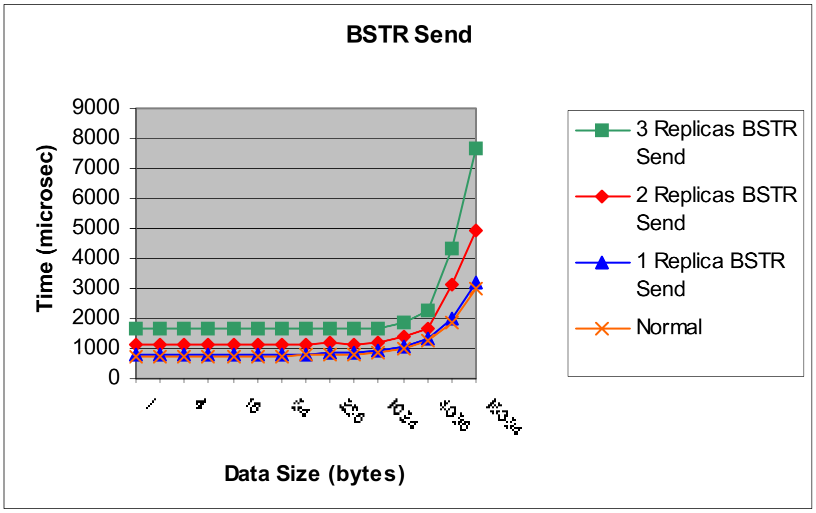
II. Multi-DCOM prototype Provides Location-Independent Performance Transparently

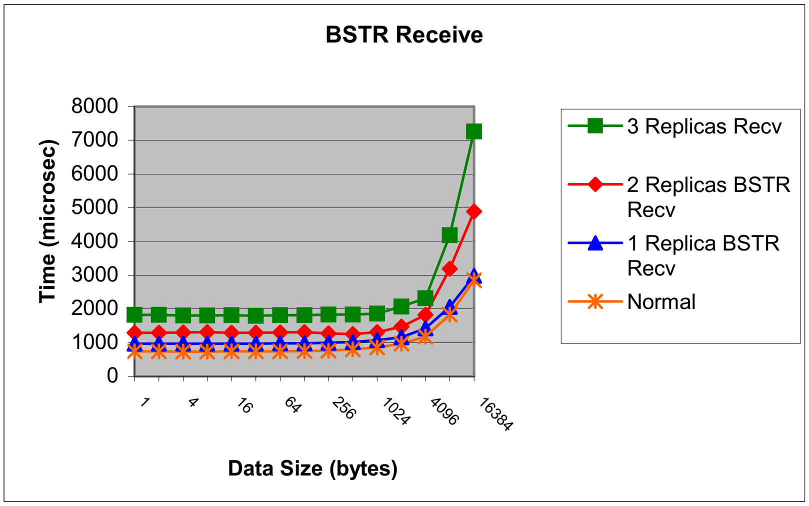
We completed the design, implementation and evaluation of a multi-DCOM prototype. We briefly describe the implementation and performance of our multi-DCOM prototype here. In order to provide support for seamless migration and redundancy to component applications designed to standard component interfaces, we are designing a multi-DCOM layer which provides transparent replication. This layer allows standard COM applications without change to do basic object replication, providing a platform for the integration of Agile Object component attributes such as migration and some types of fault-resilience. Significant technical design challenges include transparent integration and dealing with multiple return values.

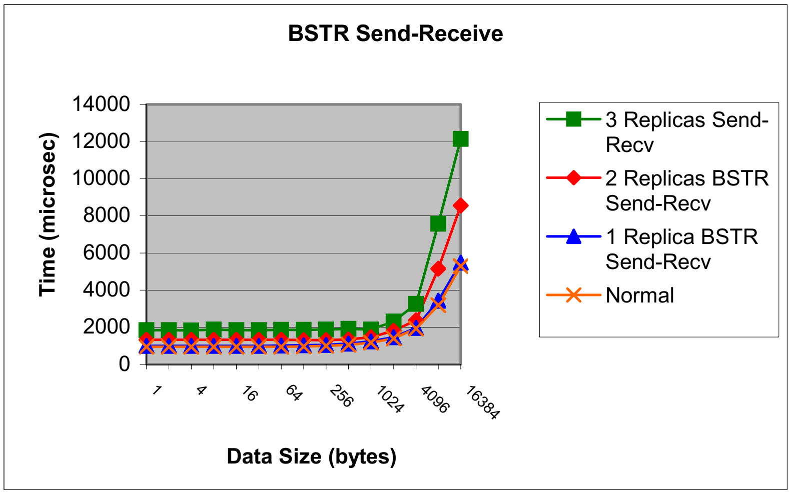
Transparent Integration Achieving transparent integration requires faithful implementation of COM interface semantics and DCOM call semantics. Since there is only one reference implementation of these semantics (that in Microsoft's Windows OS products), we are exploring an approach based on dynamically-linked library (dll) substitution and interposition of a multi-DCOM management layer. We have designed such an interception scheme which uses registry modifications and application initialization calls to specify which COM interfaces employ replication and have been exploring the internal architecture of the Windows source code to understand how we can best support the replicated invocations. Because DCOM is based on COM which in turn evolved from OLE, the software implementation structure is rather complex and non-modular. We have designed a simple scheme which provides a general enough interception capability for our needs.

Multiple Return Values One challenge in integrating replicated distributed object invocations is the handling of multiple return values. First, multiple return values must be handled correctly. Second, depending on the use of the replication, the multiple return values may be handled or combined in a different fashion. Our scheme must provide a flexible framework for doing this. Finally, it is desirable to support these capabilities with little or no disruption to the program source code. We have met these challenges by designing an "iterator" based interface. The default instantiation of the multi-DCOM system uses standard single caller and callee interfaces, and combines the results. However, replication aware interfaces can activate and iterate over responses, allowing arbitrary operations over the multiple values (such as detection of replica failure). This can require modest application change, or can be hidden in yet another layered component, allowing some replication awareness to be integrated into an application while using unmodified binary components.

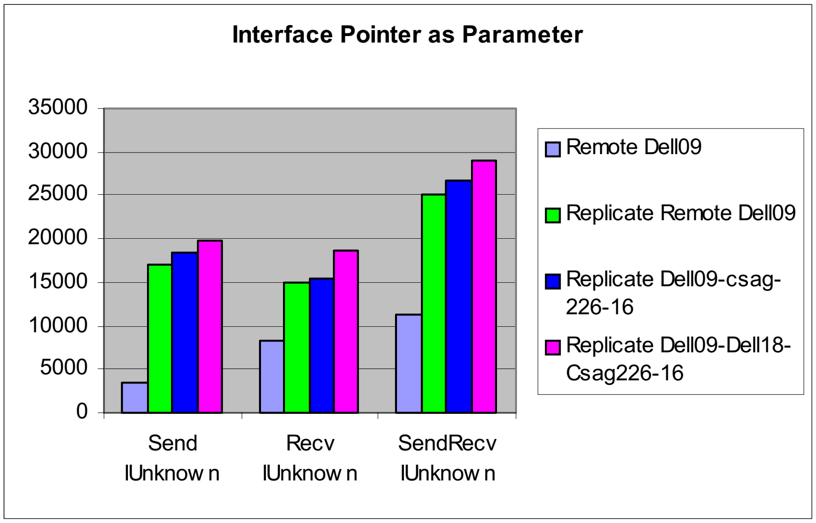
The system was implemented and tested on a legacy DCOM application which shows employee data from a shared database.   The following graphs demonstrate fast that the multi-DCOM system works, and it capabilities and performance. First the following graphs characterize the cost of using multiple replicas with a BSTR (byte array) argument:







Note that all of the y-axes are in microseconds! As can be seen from the graphs, the cost of using replicas is quite low in these systems (a fraction of a millisecond), until very large arguments are used, and the limited network bandwidth (these test were taken on a 100Mbit ethernet) becomes a constraint. The following graphs show the cost of replicas for a more complex marshalling case – a DCOM interface pointer – where the operation involves multiple network round trips, reference counting, and complex data structure traversal.



This data shows the cost of first, a remote DCOM invocation, then a replicated remote invocation, and then replication over multiple remote machines. As can be seen from the magnitude of values on the y-axis (again microseconds), the main point is that the cleaned up buffer management and pointer handling required to support replication significantly increases the cost of this case. After the first cost is paid, however, the incremental cost of adding additional replicas is small.

III. Elusive Interfaces Provides Low-overhead Invocation Protection

Elusive interfaces posits the use of changing the encoding of RPC invocations which transit the network to hide the application state. By using high level transformations on the data (rather than bit-level cryptography), reasonable protection can be provided without high computational overheads. That is, in a fashion compatible with high speed networks.

Interface Elusiveness (a.k.a. HIPIP) allows applications to have no fixed external or internal interfaces, giving electronic attackers no fixed target. The basic idea behind interface elusiveness is logical extension of ``wrappers", but mutating interfaces provide a dynamic wrapping capability. Components not designed for interface elusiveness can be automatically wrapped and enhanced. The component object middleware (Agile Objects) provides increased robustness independent of the design and implementation of the component software -- no effort on the part of application implementers is required. The component object middleware manages the online interface mutation and changes. A full exposition of this approach can be found in “HIPIP: High Performance Invocation Protection”, PhD thesis of Kay Connelly at the University of Illinois Department of Computer Science.

HIPIP Approach: While it would be ideal to strongly encrypt each remote method invocation, we have already discussed that this is not always possible for performance reasons. Instead, our approach is to make all of the method invocation messages going to a particular component look the same. If any given message could possibly invoke any of the methods on the component, it will become more difficult for an intruder to determine which method is actually being invoked.

Our results show that to transform the method invocation messages in such a fashion, it is possible to embed byte-level operations within the RPC layer of the network stack. Large overheads (which translate to prohibitive latencies for applications) are avoided in a variety of ways:

- ∉ Byte-level operations are used, which avoids expensive bit twiddling.
- ∉ Unnecessary buffer copies, which have been shown to be detrimental in high-performance messaging layers, are eliminated.
- ∉ The existing marshalling infrastructure within the RPC layer is utilized.
- ∉ Algorithms are pre-computed wherever possible, lowering the latency experienced by the message.
- ∉ Key-exchanges are performed in the background and in advance, making it less likely that a message will have to wait on a key exchange.

Since it is anticipated that an attacker can eventually decode the messages, HiPIP reconfigures itself with a new key *before* the attacker could possibly gain enough plaintext/ciphertext pairs to determine HiPIP’s internal state. An analysis of the number of plaintext/ciphertext pairs, along with an analysis of the covertime is provided in Connelly’s Ph.D. thesis.

HIPIP Design: For HiPIP, we design a security mechanism that is inserted between the components and the network, as shown in Figure 1. In essence, HiPIP acts as an interceptor. In order to initialize the security mechanisms on either side of a communication channel, HiPIP must perform some type of secure key exchange.

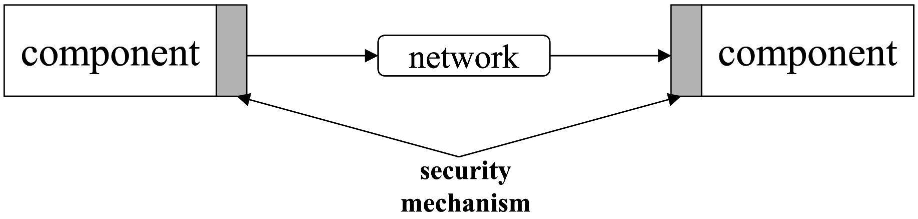


Figure 1: Placement of our HiPIP security mechanism.

Figure 2 gives the conceptual design of HiPIP. On the sending side, the plaintext of the remote method invocation is sent into HiPIP. First, the method identifier is transformed using the **offset translator**. Then, data pads are added to the message using the **padder**. Finally, the bytes in the message are moved around using the **byte transposer**. The output from the HiPIP security mechanism is sent over the network.

On the receive side, the inverse operations are performed to retrieve the plaintext. In addition to undoing the operations, the padder and the offset translator have a probe detection mechanism which allows them to determine when a message is likely to have originated from an intruder who does not have full knowledge of the internal state of the HiPIP security mechanism.

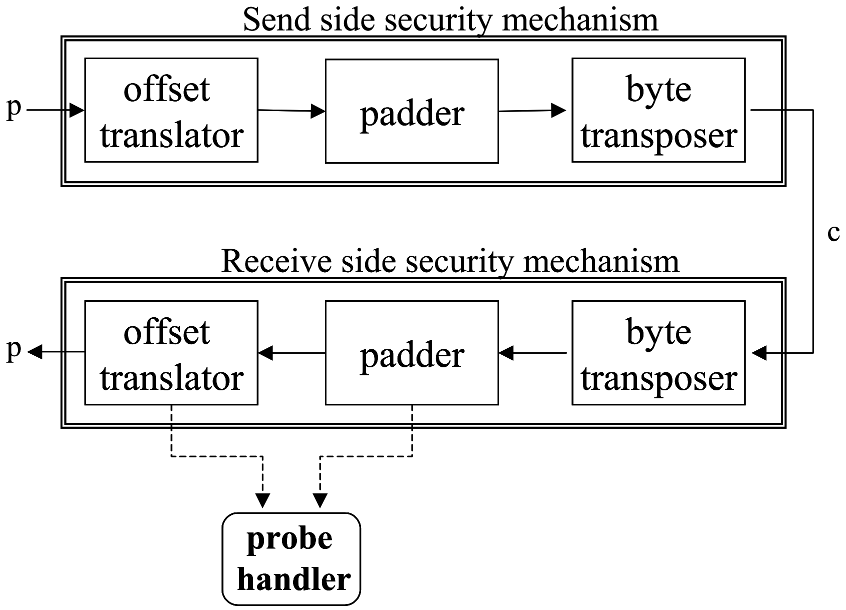


Figure 2: HiPIP design.

HIPIP Implementations and Performance: We describe the capabilities of the Elusive interfaces implementations, including the key challenges it exposed in the Elusive Interfaces security technique. We designed and implemented an Elusive Interfaces prototype based on the Manta High Performance Java RMI system. The Manta high performance RPC system achieves RMI invocations as fast as runtime, exposing the critical differences in security schemes.

While Manta does not come with DES built in, we added 64 bit-key, single DES within the Manta framework so that we could make a fair performance comparison between DES and HiPIP. Similar to the HiPIP implementation, our DES modifications change the RMI data flow so that once the message is marshaled into the message buffer, the DES library is invoked to encrypt the message buffer. On the receive side, a special DES handler is invoked by the Panda subsystem. The DES handler decrypts the message buffer using the DES library and then passes the buffer onto the server-side stub for normal processing.

Figure 3 shows the DES latency along with the unmodified Manta and our HiPIP implementation. Unlike the other performance graphs in this chapter, the y-axis is a logarithmic scale. For a null RMI, Manta and HiPIP latencies are 150 and 156 microseconds, respectively. The average DES latency for a null RMI is 29,682 microseconds. In general, the average DES latency is 2-3 orders of magnitude greater than HiPIP. Regression analysis determines that DES's per byte overhead is 1733 microseconds[1]. In contrast for HiPIP, the per byte overhead is 0.38 microseconds, approximately 4500 times smaller.

[1] The per message overhead is insignificant as compared to the per byte overhead, thus the regression analysis forced the per message overhead to zero, with a resulting r-square value of 0.999.

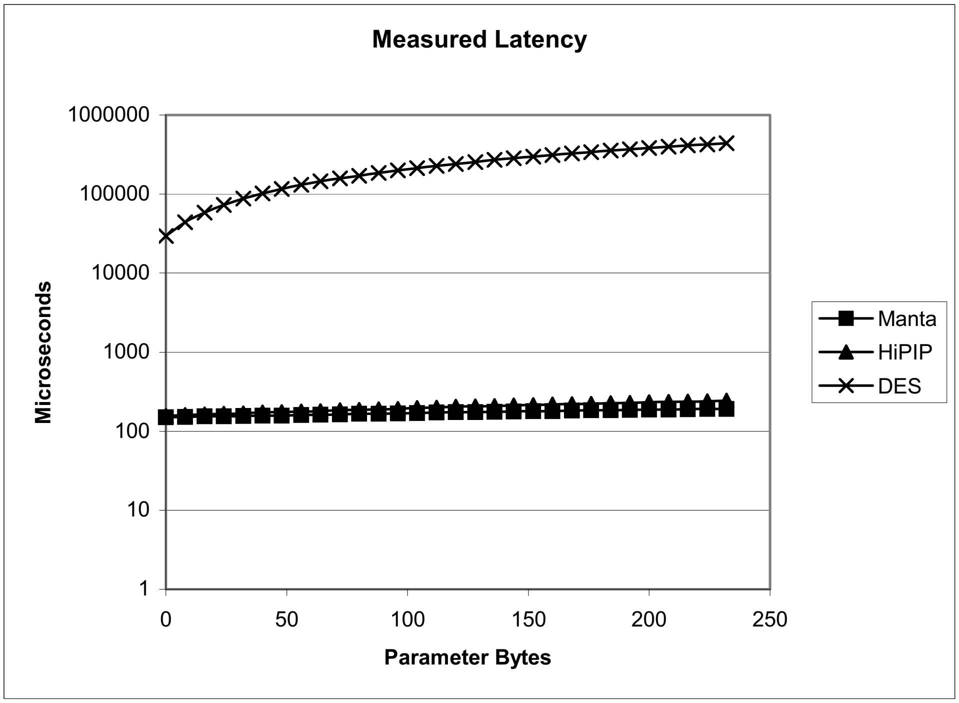


Figure 3: DES latency comparison (logarithmic scale).

This prototype demonstrated the principle behind interface elusiveness, and is a realization of a primitive Elusive interfaces capability. The implementation has been empirically characterized, and despite the typical limitations attendant on a first generation prototype, demonstrates good performance compared to typical encryption techniques. In particular, the graphs below illustrate the performance scaling properties of Elusive interfaces versus message parameter size and number of parameters when compared to traditional encryption techniques.

These results show that Elusive interfaces can be competitive with ordinary marshalling code, and therefore can be used on the fastest networks available. In addition, the relative speed of Elusive interfaces to traditional encryption is dramatically faster, demonstrating that there is a significant potential performance benefit to Elusive interfaces in some performance regimes. Future reports will include deeper studies and more sophisticated Elusive interface studies.

Application Performance: To assess the performance of HiPIP on larger applications programs, we ran three applications with different interface complexities and communication patterns using HiPIP, DES and plaintext. ASP is an all-pairs shortest path algorithm which uses Floyd-Warshall. FFT is computed using the transpose algorithm, and SOR is a red-black alternation approach. Full descriptions of these application kernels can be found in Connelly’s Ph.D. thesis. ASP uses totally-ordered broadcast, FFT has all-to-all communication, and SOR has neighbor-only communication. All three applications were obtained from the Manta developers. While we recompiled the applications with our HiPIP compiler options, we did not modify the source code.

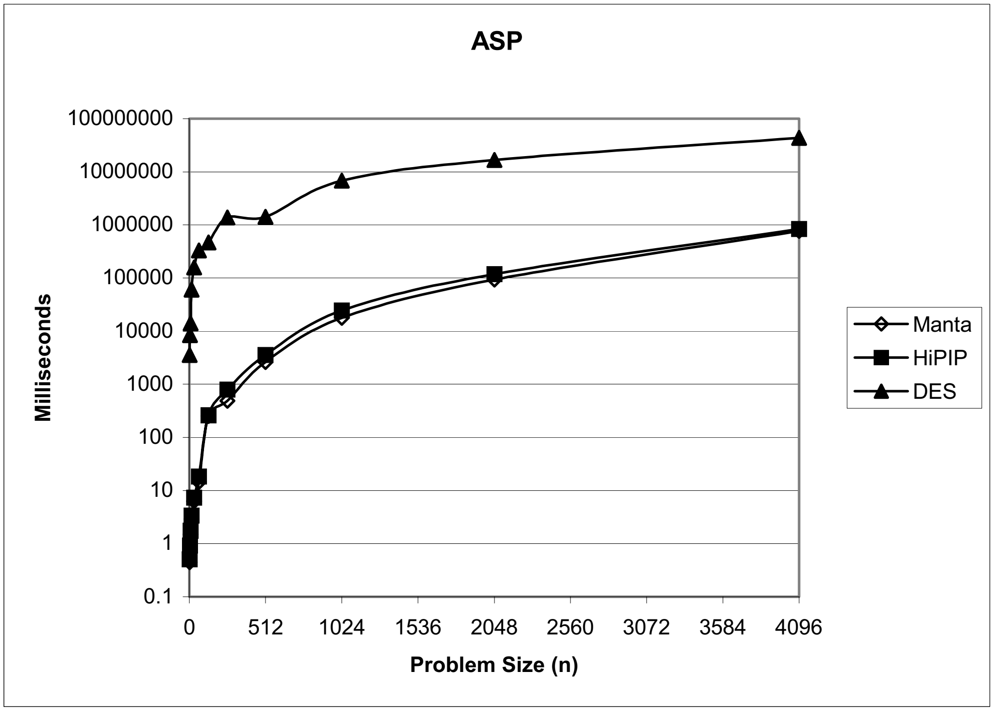


Figure 4: ASP performance.

Figure 5 gives the FFT application latency running on 2 nodes with a matrix size of 8, 10 and 12. The HiPIP performance is within 2% of Manta; whereas DES is over 9 - 250 times slower than Manta.

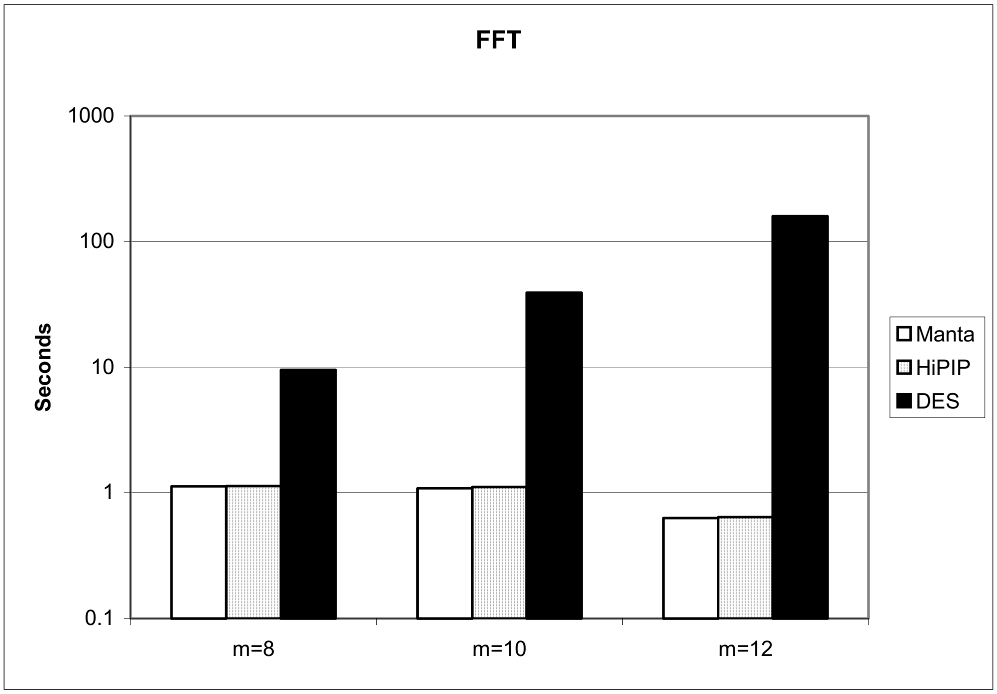


Figure 5: FFT performance

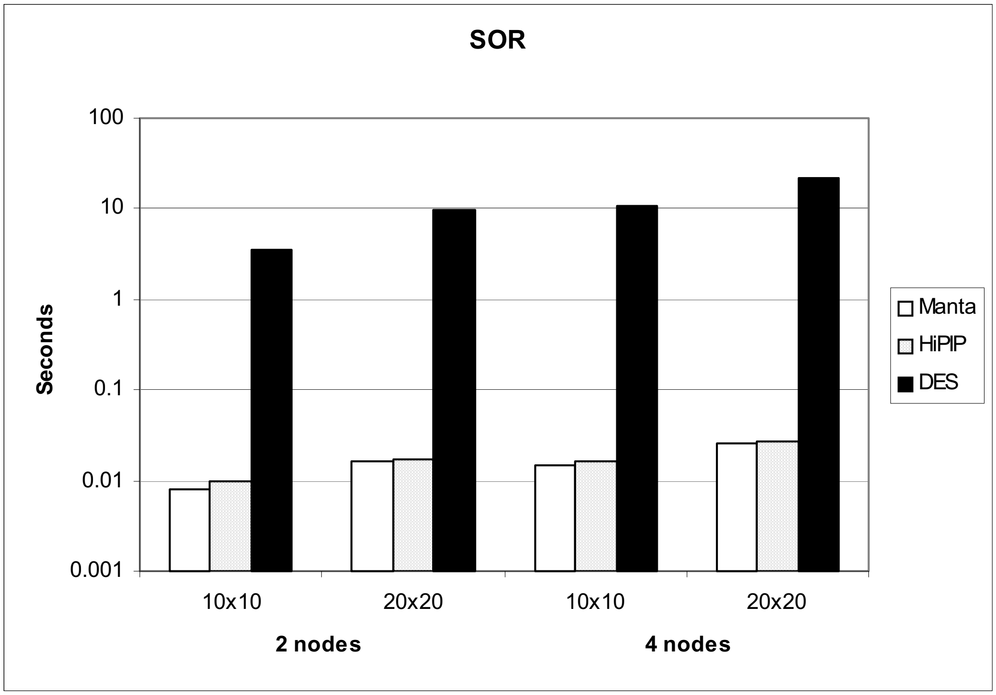


Figure 6 gives the performance of the SOR application running on 2 and 4 nodes for 2 different problem sizes (10x10 and 20x20 matrices). HiPIP incurs and average overhead of 11% of the base Manta latency. DES, on the other hand, is 400 - 800 times slower than Manta and HiPIP.

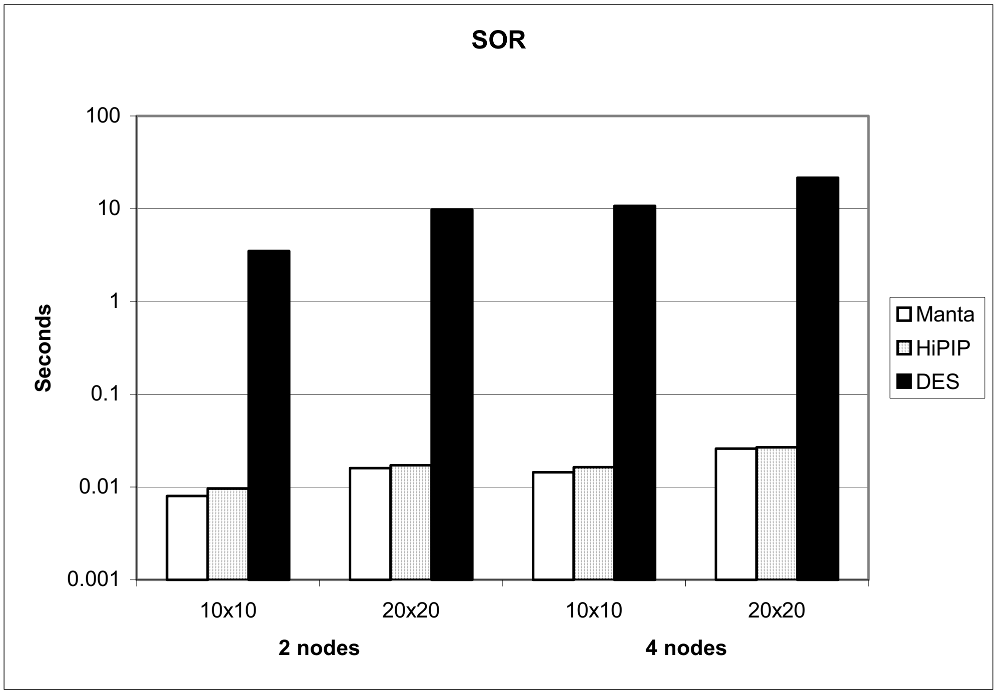


Figure 6: SOR performance.

The three parallel applications each used different communication patterns.   With all three applications, HiPIP performance is the same order of magnitude as Manta; DES, however, is consistently a 2-4 orders of magnitude slower than Manta and HiPIP.

IV. Open Real-time systems for Agile Objects – REALTOR

The objective here is to develop, implement, and demonstrate a *real-time capable* Agile Object System (this work was done primarily by the team at Texas A&M). By real-time capable it is understood that requests to agile objects continue to satisfy timing guarantees despite overhead due to migration and reconfiguration: Timing requirements are satisfied before, during, and after migration and reconfiguration. We have accomplished *(a)* the selection, evaluation, and appropriate adaptation of a real-time capable infrastructure for agile objects (Java-RTSJ with extensions for real-time RMI), *(b)* support for real-time component migration, and *(c)* a framework for distributed light-weight admission control.

In summary, we achieved most of the laid out goals: (a) the real-time infrastructure (Real-time Java, with real-time schedulers and real-time extensions to RMI) was put in place; (b) the support for real-time migration was implemented and successfully demonstrated; (c) scalable and light-weight admission control was designed, implemented, and successfully demonstrated. In the following, we describe in detail the specific accomplishments.

Distributed Real-Time Architecture: We have developed and refined a distributed real-time architecture that provides a framework for resource protection and light-weight admission control. This infrastructure is based on real-time Java, with real-time schedulers and real-time extensions to RMI. Our architecture consists of a *global resource manager* (realized in form of the REALTOR resource manager), *local resource manager* (realized in form of Access Control modules), and *guaranteed-rate scheduling* at the nodes. Guaranteed-rate schedulers (in our case we use a Total Bandwidth scheduler) are more expensive than simpler (e.g. static-priority) schedulers. They have the benefit, however, that portions of the CPU capacity can be safely allocated to individual workload streams, thus allowing for "Virtual CPUs" to be defined and assigned to workloads. This in turn significantly simplifies both resource discovery and allocation and admission control and component migration.

Light-Weight Distributed Admission Control: We have developed a light-weight admission control framework and implementation within the Agile Objects distributed real-time architecture. This system which provides light-weight distributed admission control was successfully developed, integrated into the AO system, and tested and evaluated.

The light-weight distributed admission control consists of two portions: *(a)* admission control mechanism, and *(b)* scalable resource discovery.

(a) Light-Weight Admission Control Mechanism: The admission controller (Figure 7) is invoked either during resource discovery, or during component creation or migration.

The admission control *protocol* (interaction with peer controllers, with the resource managers, and with the underlying creation and migration modules) is defined in the Admission Control module. The admission control *logic* is defined by an extensible set of modules that control the local resource allocation and job placement. Examples are the Job Scheduler module (guarantees real-time constraints), and the Security module (implements job placement policies). The decision to use guaranteed-rate schedulers in the AO nodes greatly simplifies both the admission control protocol *and* the admission control logic, since admission decisions rely on utilization levels only.

(b) Scalable Resource Discovery: We developed a new resource discovery protocol, REALTOR, which is based on a combination of pull-based and push-based resource information dissemination. REALTOR has been designed for real-time component-based distributed applications in very dynamic or adverse environments. REALTOR is characterized by low-overhead communication, soft-state operation (i.e. no hard state about resource availability in the network has to be maintained), completely idempotent operation (communication failures or inaccurate data have either only minor and temporary effects, or no effects at all). Simulation studies show that under normal and heavy load conditions REALTOR remains very effective in finding available resources with a reasonably low communication overhead. Our evaluation shows that REALTOR *(a)* effectively locates resources under highly dynamic conditions, *(b)* has an overhead that is system-size independent, and *(c)* works well in highly adverse environments. REALTOR is described in detail in a paper presented at WPDRTS-2003 in April 2003.

<u>Real-Time Component Migration</u>: We have develop a low-disruption component migration mechanism within the AO distributed real-time framework which allows for continuous real-time guarantees during component migration. This mechanism was successfully realized and demonstrated as part of the AO File Server Demonstrator. Real-Time migration is achieved through a combination of two mechanisms: *(a)* migration-aware admission control, and *(b)* pro-active resource discovery in REALTOR.

(a) Migration-aware Admission Control: The bandwidth preserving schedulers in AO nodes allow for appropriate resource slices to be reserved for future migration of AO components. This is integrated into the admission control of every newly created or migrated AO component. This works under the assumption that resources available for migration have been identified before the migration has to take place. Otherwise, the location of appropriate resources in the network may cause excessive delays and missed deadlines. We address this through appropriate resource discovery.

(b) Pro-active Resource Discovery: Successful real-time migration depends on low-latency location of available resources. In a number of (simulation) experiments, we have demonstrated that REALTOR provides a scalable and effective means to direct the migration mechanism to available resources in the network, and so to achieve real-time migration even for high utilization levels of resources in the network. Naturally, REALTOR's effectiveness decreases (rates of migrations missing their deadlines increase) when utilization levels are exceedingly high. Separate mechanisms must be used during system deployment and admission control of new components to ensure that sufficient unclaimed resources remain in the network for REALTOR to be effective. (This gives raise to a trade-off between overhead in resource discovery schemes like

REALTOR and the overall utilization level of resources, which we are planning to further investigate.)

Prototype Implementation of Admission Control Framework and Integration into AO System: We successfully built a prototype implementation of this framework within Java-RTSJ/RMI to empirically demonstrate the viability of our approach. This prototype integrates the admission control and migration mechanisms to achieve location elusiveness. The major contributions here include: (a) Real-time extensions to Java, (b) Real-Time Migration, (c) Resource Discovery and Allocation (REALTOR). These contributions will be described in detail:

(a) Real-time Extensions to Java: As the underlying platform, we selected the Real-Time Specification for Java (Java RTSJ). RTSJ had to be extended in three ways to make it applicable within the AO project:

(a.1) RTSJ compliant veneer for Windows-NT: The only RTSJ reference implementation was available from TimeSys to run over their Linux-RT kernel. In order to support UCSD's efforts, we developed a RTSJ-like environment to run over Windows-NT. This version is largely RTSJ compliant, except for memory management and asynchronous event handling.

(a.2) Extensions to Real-Time Scheduler: In its basic form, RTSJ provides a simple static priority scheduler. We developed an earliest-deadline-first (EDF) scheduler for RTSJ. Based on this EDF scheduler, we developed a Total Bandwidth (TB) scheduler. TB is a guaranteed-rate scheduler, and provides a "virtual CPU" to each of the threads in the system. This greatly simplifies both the admission control in real-time AO and the realization of the real-time extensions to RMI.

(a.3) Real-Time RMI: RTSJ in its current form has no support for distributed real-time computation. In particular, there is no support for real-time RMI. We integrated the Java RMI classes with RTSJ by allowing the RMI runtime threads to be executed under control of the AO real-time scheduler. These modified Java RMI classes create real-time worker threads for the RMI invocations, and so guaranteed-rate scheduling can be applied. Given the server-centric approach used in real-time AO, very little overhead information (such as timeliness or priority-inheritance parameters) needs to be carried with the remote invocation messages. Particular attention is given in the admission control module for *blocking due to self-suspension* during remote invocations. This form of blocking is inherent to remote invocation systems, must be taken into account during worst-case delay analysis, and thus leads to reduced resource utilization. In order to reduce the worst-case effects of this form of blocking, this led us to investigate the use of *asynchronous RMI* in Java. This study, however, is not complete.

(b) Real-Time Migration: We support real-time migration for AO components based on the Admission Control and Resource Allocation modules. The architecture supporting real-time migration is illustrated in Figure 7.

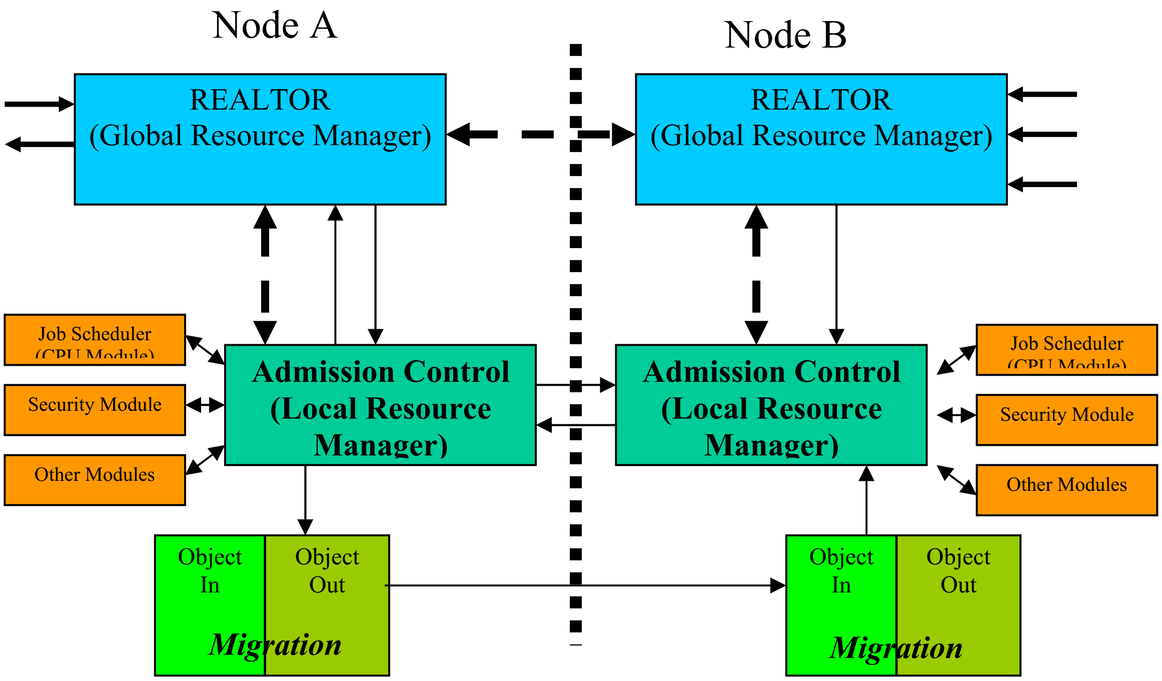
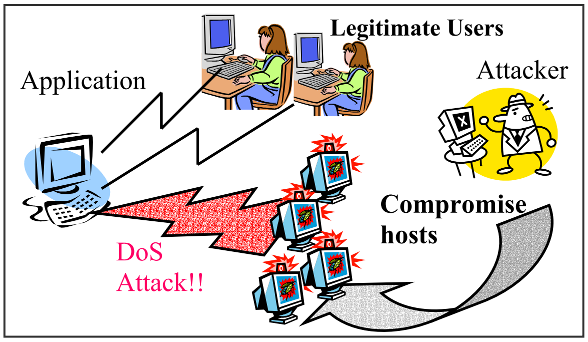

Node A
Node B
REALTOR (Global Resource Manager)
REALTOR (Global Resource Manager)
Job Scheduler (CPU Module)
Security Module
Other Modules
Admission Control (Local Resource Manager)
Admission Control (Local Resource Manager)
Job Scheduler (CPU Module)
Security Module
Other Modules
Object In
Object Out
Migration
Object In
Object Out
Migration

Figure 7: Support for Real-Time Migration in Real-Time Agile Objects

(c) Resource Allocator (REALTOR): REALTOR has been implemented and tested. REALTOR nodes interact with peer nodes to discover resources and manage an approximate picture of resource availability in the AO system. They interact with the Admission Control modules on the local AO node to allocate and manage resources. The interaction between REALTOR and the Admission Control module is depicted in Figure 7.

V. Denial of Service Resistance Using Agile Objects

We began exploration of how to tolerate, and to what degree it is possible to tolerate Denial of Service attacks in Agile Objects systems which implement "location elusiveness". Denial-of-service (DoS) attacks have been a major security threat to Internet applications. Since 1998, there have been several cases of large-scale distributed DoS attacks, during which popular sites such as Yahoo! and Amazon were shut down, and an important government website was forced to move to a different location. These attacks have serious economic and political impact, and may even threaten critical infrastructures and national security.



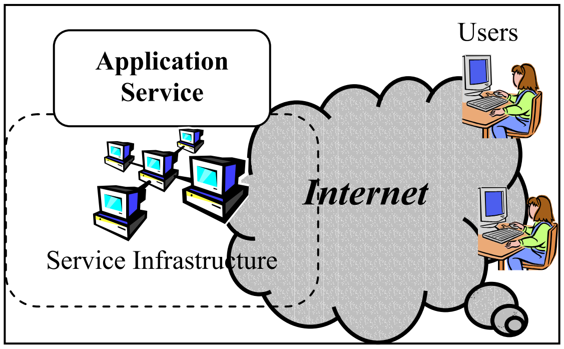


Figure 8. Denial-of-Service Attack **Figure 9. Example of Internet Application**

As shown in Figure 8, a typical DoS attack has two stages. In the first stage, attackers compromise many hosts in the Internet and install "zombie" programs. In the second stage, the attackers control these zombie programs to attack the victim. According to the method of attack used in the second stage, DoS attacks can be categorized as *infrastructure level* or *application level attacks*. Figure 9 shows a typical Internet application deployment. The application service runs on a resource pool of interconnected hosts; users access it via the Internet. *Infrastructure level attacks* attack the resource pool directly, for example, by sending packet floods to saturate the victim's network. *Application level attacks* cause denial-of-service by requesting significant amount of workload or by exploiting weaknesses on the application.

The fact that most Internet applications are publicly accessible makes them easy targets for *infrastructure level DoS attacks*.

Researchers are studying the use of overlay networks to tolerate DoS attacks on Internet applications. We consider an overlay proxy network approach to tolerate *infrastructure level attacks* on publicly accessible applications. The key idea is to hide the application behind a proxy network, which itself is embedded in a network of a huge number of Internet hosts. The proxy network and applications use the Internet hosts as a resource pool; users access the applications via some edge proxies with known IP addresses. The proxy network can dynamically reconfigure so that attackers cannot easily locate the application, preventing the launch of *infrastructure level* DoS attacks. In addition, the applications are able to be moved amongst the hosts, separating them from dependence on a particular infrastructure, allowing them to tolerate infrastructure attacks. However, we focus on proxy network reconfiguration here as a primary way of system reconfiguration.

In this paper, we build a formal model and use it to study the effectiveness of overlay networks to tolerate DoS attacks. More specifically, subject to the formal model, we characterize how quickly resources can be compromised and the effectiveness of policies such as intrusion detection triggered recovery or a simple periodic system reset. We also characterize the difficulty for attackers to discover the location of the application. Applications of these models to several system scenarios yield the following novel conclusions:

∉ Intrusion detection-triggered recovery strategy is insufficient to avoid resource depletion.

∉ True-positive rates of intrusion detectors have more impact on resource availability than detection speed.

∉ System reconfiguration techniques such as random proxy migration can effectively prevent attackers from discovering applications' locations.

∉ Overlay network topology is critical; richly connected topology may reduce a proxy network's effectiveness in resisting attacks.

A more complete exposition of these results can be found in the relevant papers by Wang, Liu, and Chien listed below.

III. Students and Staff

The following students were supported on this contract:

UIUC Students

∉ Oolan Zimmer, UIUC

∉ Geetanjali Sampemane, M.S. 2000, continuing PhD Student

∉ Sudha Krishnamurthy, PhD 2002

∉ Kay Hane Connelly, PhD 2003, now Assistant Professor, Indiana University

∉ Luis Rivera, M.S. 2000, continuing PhD student

UCSD Students

∉ Kiran Tati (continuing PhD student)

∉ Xin (Paff) Liu (continuing PhD student)

∉ Huaxia Xia (continuing PhD student)

∉ Ju (Tony) Wang, M.S. 2000 (continuing PhD student)

TAMU Students

∉ Byung Choi, PhD 2002, now Assistant Professor, Michigan Tech University

∉ Sangig Rho, continuing PhD student

The following staff were supported on this contract:

∉ Philip Papadopoulous (now Director of Cluster and Grid activities at SDSC)

∉ Mason Katz (now SDSC cluster team leader)

∉ Alex Olugbile

A number of the students and staff have gone on to outstanding industrial and academic opportunities. A number of the students continue at UCSD, and will finish their PhD's within the year. Alex Olugbile continues with the Concurrent Systems Architecture Group at UCSD and is a leading technical contributor for a wide range of research projects.

IV. Software and Technology Transfer

The Agile Objects project has successfully pursued a multi-channel approach to technology transfer involving publication, demonstrations, and software availability.

Publish Papers: As documented elsewhere in this report, we have published numerous papers in leading conferences which document the advanced capabilities and key technologies developed by the Agile Objects project.

Demonstrate Software: As documented elsewhere in this report, we have demonstrated a series of working prototypes. These projects constitute an embodiment of the technical advances developed by the Agile Objects project.

The Agile Objects project has produced several major demonstrations and software releases involving demonstrating each of the capabilities of the Agile Objects middleware in a range of application scenarios. The major software demonstrations, functionality, and release dates are summarized below.

DCOM Transparent Interception System (UCSD Site Visit, May 2000)

This system provided transparent interception of DCOM invocations thru modification of a custom transport provider in the Microsoft Windows NT operating system software. The demonstration took an unmodified DCOM application, a corporate information program, and interfaced it to two distinct data base server systems. Upon external command, the client was diverted by the intercepting Agile Objects middleware to the second database server system. No interruption in service or loss of session occurred at the client. Thus, an unmodified binary application was transparently intercepted and diverted, showing the potential for Agile Objects properties to be provided to legacy distributed and component applications without requiring application redesign or even recompilation. Details of this system and demonstration can be found in Ju (Tony) Wang's MS thesis.

Agile Objects Streaming Video Demo (PI Meeting, August 2002)

This system provided a tangible demonstration of a mobile file server application built from Agile Objects. This file server was used to provide video files to a streaming media server and web server. The mobility of the file server demonstrated the feasibility of the Agile Objects paradigm for back-end server applications, and the demo showed that continuous service can be achieved for clients of mobile server applications using Agile Objects name services and rapid updates.

The real-time extensions to the Agile Object framework were also demonstrated as part of the AO File Server Demonstrator. This demonstration successfully illustrated the functionality of the real-time scheduling support and the real-time RMI capabilities on top of a standard Java platform in WindowsNT (using the partially RTSJ compliant veneer).

Image Processing Pipeline Demo (December 2002)

Supporting complex networks of Agile Objects is an important goal of the AO project as it enables larger, more complex applications to benefit from agility of many types. We built a second demonstration system which allows arbitrary networks of Image Processing operators to be configured (a complex network of distributed objects or components), and each of these components can then be migrated around the resource pool. The Agile Objects middleware maintains the connections between the objects, despite rapid migration, allowing the applications to perform with a seamless capability. The applications can thus roam across the resource environment, making them independent from any resource failures or compromises. Integration of real-time AO support into the *AO Image Processor* is on-going. We are experiencing delays with the integration of real-time AO with Linux and with the porting of a real-time capable Linux platform onto the experimentation hardware.

HIPIP system (August 2003)

The High Performance Invocation Protection System (a.k.a. Elusive Interfaces), integrates into a high performance Java RMI system (Manta) the capability to transparently change distributed object interfaces to improve application data security at little or no cost to performance. The system supports any applications written in Java using RMI to communicate which respect a “closed world” assumption (a Manta restriction, not one added by AO). This system has been used to run large applications, and to demonstrate that HIPIP can be as much as 500 times faster than traditional cryptographic approaches such as 64-bit DES. Despite this speed advantage, HIPIP can provide some important data confidentiality; though not the strength of DES. As a result, HIPIP is useful and represents a significant new capability for high speed cluster computing environments.

V. Papers

∉ Tony Wang and Andrew Chien, Using Overlay Networks to Resist Denial-of-Service Attacks (Submitted to ICDCS'04), July 2003.

∉ Tony Wang, Linyuan Lu, and Andrew A. Chien, "Tolerating Denial-of-Service Attacks Using Overlay Networks – Impact of Overlay Network Topology" (pdf), to appear in 2003 ACM Workshop on Survivable and Self-Regenerative Systems, October 2003.

∉ Kay Hane Connelly, "HIPIP: High Performance Invocation Protection", Ph.D. Thesis, University of Illinois Department of Computer Science, August 2003.

∉ B.-K. Choi, S. Rho, and R. Bettati "Dynamic Resource Discovery for Applications Survivability in Distributed Real-Time Systems", 11th International Workshop on Parallel and Distributed Real-Time Systems, Nice, France, Apr. 22-23, 2003.

∉ Tony Wang and Andrew Chien, An Analysis of Using Overlay Networks to Resist Distributed Denial-of-Service Attacks (TechReport), December 2002.

∉ B. Choi, S. Rho, and R. Bettati, Dynamic Resource Discovery for Applications Survivability in Distributed Real-Time Systems, submitted to the International Workshop on Parallel Distributed Real-time Systems 2002.

∉ Kay Connelly and Andrew Chien, Breaking the Barriers: High Performance Security for High Performance Computing, in New Security Paradigms Workshop, Virginia Beach, September 2002.

∉ B.-K. Choi: "Resource Management for Scalable Quality of Service." (PhD Thesis, Texas A&M University, August 2002)

∉ K. Connellly, L. Rivera, G. Sampemane, K. Tati, T. Wang, and A. Chien*Agile Objects: Survivable Middleware for Distributed Systems* Tech. Report, April 2001.

∉ Tony Wang, *Transparent Replication for Component-based Applications* (Master Thesis) Fall 2000.

∉ K. Connelly and A. Chien, *Elusive Interfaces: A Low-Cost Mechanism for Protecting Distributed Object Interfaces* (PDF) Submitted for publication, May 2000.

∉ O. Zimmer and A. Chien, *The Impact of Inexpensive Communication on a Commercial RPC System* UIUC Department of Computer Science Technical Report, August 1998 (Zimmer, Chien)

∉ *Performance of OmniBroker, an Implementation of CORBA 2.0* Tech. Report, February, 1998.

VI. Key Published Papers

A collection of the key published papers are included as examples of project research contributions.

∉ Kay Connelly and Andrew Chien, Breaking the Barriers: High Performance Security for High Performance Computing, in New Security Paradigms Workshop, Virginia Beach, September 2002.

∉ B. Choi, S. Rho, and R. Bettati, Dynamic Resource Discovery for Applications Survivability in Distributed Real-Time Systems, submitted to the International Workshop on Parallel Distributed Real-time Systems 2002.

∉ Tony Wang and Andrew Chien, Using Overlay Networks to Resist Denial-of-Service Attacks (Submitted to ICDCS’04), July 2003.

Breaking the Barriers: High Performance Security for High Performance Computing

Kay Connelly
Indiana University
150 S. Woodlawn Ave.
Bloomington, IN 47405
(812) 855-0739
connelly@indiana.edu

Andrew A. Chien
University of California, San Diego
9500 Gilman Drive, Dept. 0114
La Jolla, CA 92093-0114
(858) 822-2458
achien@cs.ucsd.edu

ABSTRACT

This paper attempts to reconcile the high performance community's requirement of high performance with the need for security, and reconcile some accepted security approaches with the performance constraints of high-performance networks. We propose a new paradigm and challenge existing practice. The new paradigm is that not all domains need long-term forward data confidentiality. In particular, we take a fresh look at security for the high-performance domain, focusing particularly on component-based applications. We discuss the security and performance requirements of this domain in order to elucidate both the constraints and opportunities. We challenge the existing practice of high-performance networks sending communication in plaintext. We propose a security mechanism and provide metrics for analyzing both the security and performance costs.

General Terms

Distributed Computing, High Performance, Security.

1. INTRODUCTION

Over the past decade, high performance networks of workstations have replaced supercomputers for scientific parallel computations. As these clusters have become easier to manage and use, distributed applications outside of parallel scientific codes have targeted this platform as well. Search engines, airline reservation systems and command-and-control systems are just a few such applications. The combination of low-cost and high-performance execution has made such systems desirable to a wide variety of industries. In particular, developments in user-level communication layers have enabled applications to access the raw performance of such networks. Applications achieve peak bandwidths over 1Gbps, and latencies on the order of 10 to 20 microseconds.

With the focus centered on performance, there has been little research into security for high performance systems. Much of the security-related work in the high performance computing (HPC) community addresses how to securely communicate to high-performance applications from the wide-area (i.e.: how to retrieve remote data sets or how to securely start a remote high-performance application) [1, 6]. But, beyond simple logins and access rights associated with those logins, there are few security mechanisms being regularly employed within the high-performance clusters, themselves. In terms of the communication going over the high performance network, the standard practice is to have *no* security. All data is sent in plaintext. The main goal is to keep the communication overhead to a minimum. Grafting an existing encryption mechanism onto the communication path is not seriously considered due to the relatively high overheads. A typical symmetric key encryption algorithm incurs an overhead of *milli*seconds, which is two to three orders of magnitude greater than the network latency in high-performance networks. Using such a mechanism would take the "high performance" out of HPC. Now that industry and the military are seriously pursuing high performance clusters as an environment to run their distributed applications, the HPC community must revisit the issue of security.

Distributed components are quickly becoming the programming model of choice for distributed high performance applications [16]. In this type of model, the functionality of the application is encapsulated in multiple components and spread over the network. In order for the application to make any forward progress, components must interact with other components via remote procedure calls (RPC). Thus, the state of the execution of the application can be pieced together with these RPCs.

One noticeable affect of low-latency communication is that the ideal balance of computation and communication changes dramatically from traditional TCP/IP over Ethernet. On the slower networks, a component must compute a lot and communicate rarely in order to achieve its peak performance. If it doesn't have enough computation to keep it busy while waiting on the results from an RPC, then it becomes idle waiting on the network. In the high performance domain, communication is many orders of magnitude faster. Thus, components can have much less computation, and still not block on the network. This results in applications that have many fine-grained components (as opposed to fewer, larger components). Finer-grained applications have more RPCs, making a more detailed state-reconstruction possible.

There are two important security attack scenarios for high performance component applications. The first is for an attacker to send RPC messages to various components in order to change the execution of the application. The attacker may not need to reconstruct the current state of the application in order for such an attack to succeed. Since RPCs are currently sent in plaintext with no authentication mechanisms, this attack is feasible as long as the locations of the components are accessible. The second scenario consists of an attacker eavesdropping on the communication and determining when the application is in a vulnerable state. The attacker can then attack the application, another application, or use the information to gain an advantage in the real world .

The contributions of this paper include:

- A discussion of specific security and performance needs of high performance applications.
- An approach for protecting tightly-coupled, high-performance, component communication.
- Definition of security and complexity metrics to analyze this approach.
- A characterization of the security achieved by this approach. For a modest sized component, this approach provides a brute-force search space of 10^{28}. A known-plaintext attack requires at least 20 plaintext/ciphertext pairs.
- A proof-of-concept prototype that adds less than 10% to the message latency.

Section 2 describes the shift in the way we must think about security for high-performance systems. Section 3 gives one possible approach to satisfying the security and performance requirements of this domain and introduces the metrics we use to analyze the approach. We apply these metrics to three particular security techniques in Section 4. Section 5 describes an initial prototype with performance numbers which demonstrate that this approach is promising in terms of performance. Finally, we describe some related work in Section 6, and conclude in Section 7.

2. PARADIGM SHIFT

When looking at security for specialized domains such as high-performance component applications, we cannot naively apply existing security solutions without potentially sacrificing the benefits of that domain. Instead, we must evaluate the needs of the system. There are security needs, but there are other needs, such as performance, reliability and usability.

In the case of HPC, the driving force is performance. Existing security mechanisms simply incur too much overhead for them to be adopted by the HPC community. Thus, we have an additional restriction on security mechanisms, in that they must have a low overhead. “Low overhead”, of course, is a fuzzy term. For now, let it be sufficient that the overhead incurred by the security mechanism must be the same order of magnitude as the latency of the message sent in plain text. In the case of 100 Mb switched Ethernet sending small messages, this means that the security mechanism may incur an overhead up to 100 microseconds in order to satisfy the performance constraint.

Now, let us turn to the security needs of high-performance component applications. The bulk of the communication in this type of application is temporary data or information related to the control flow of the application. For example, in the case of scientific, parallel applications, the data traversing the network might be intermediate values in a computation. In the case of a command-and-control application, the data may consist of sensor-values or simple Booleans to enable and disable various resources. The risk of the communication being exposed is not that the data is valuable, but that the data may indicate that the application is in a weakened state, making it vulnerable to a specific attack. It does not matter if an attacker is able to determine the current state of the application in a few hours, minutes or seconds, as the application will have moved on to another state. This is an important change in perspective: long-term forward security is not the ultimate goal. The goal is to protect the data long enough for the application to change state, and to do so with low overhead.

While any sensitive data which needs long-term forward security must use a traditional encryption mechanism, the bulk of the communication in our target applications consists of these intermediate, or short-term, values; and thus, they have a shorter cover time than traditional data. This gives us a new opportunity when designing a security mechanism. In order for an application programmer to be able to determine if the cover time is long enough for their particular application, it will be necessary to precisely quantify the cover time provided by any proposed mechanism. In the most naïve attack, the cover time is roughly proportional to the size of the brute-force search space. In a more sophisticated attack on the state of the security module, we must determine the frequency with which the module must be reconfigured with a new secret key. This frequency must be low enough that the overhead of transmitting the keys does not dominate.

A key question for a given application is: how long does the cover time need to be? The security requirements depend on the frequency of the state changes. For a loosely-coupled application, communication (and thus state changes) are infrequent, necessitating a longer cover time. For tightly-coupled applications, communication and state changes are frequent, requiring a much smaller cover time. In essence, there is a range of communication patterns and security requirements.

Previous work [7] provides a high-performance communication library which allows the application programmer to turn communication security (DES) on and off. We believe, however, that more than two modes are needed in order to get the HPC community to actually *use* the provided security. For example, triple DES, which incurs one-way overheads on the order of a few milliseconds for 4k messages, could be used for loosely coupled applications. Single DES, which incurs overheads on the order of 500 microseconds could be used for applications which are somewhere in the middle of the tightly-to-loosely coupled continuum. And finally, instead of the plaintext option, another mechanism should be available which incurs virtually no overhead and provides very short term security for tightly coupled applications.

There already exist security mechanisms for the medium to loosly-coupled applications. The rest of this paper explores one possible approach for a security mechanism specifically designed for tightly-coupled components.

3. APPROACH

Existing encryption mechanisms apply operations such as substitution and transposition in an iterative fashion on the data. For every iteration, data is read from a buffer, transformed in some way, and copied to another buffer. Analysis of messaging layers shows that buffer copies are one of the major sources of overhead to avoid [3, 9]. Indeed, zero-copy messaging layers have become the accepted norm in the HPC community.

Thus, when developing a security mechanism for tightly-coupled, high-performance applications, it is necessary to avoid buffer copies whenever possible; making an “iterative” approach undesirable.

Instead, our approach applies traditional security techniques such as transposition, substitution and data padding while the message is being marshaled onto the wire. We apply these operations on the primitive data types (i.e, bytes and words) in the RPC marshalling layer. This allows us to avoid all buffer copies, and to capitalize on the marshaling infrastructure that already exists, adding what we anticipate to be a modest amount of overhead.

In addition, much of the computation in the techniques we propose can be done before the message becomes available from the application. This allows our system to pre-compute the more time-consuming algorithms during any CPU idle time, significantly reducing the communication latency experienced by the application.

3.1 Metrics

In order to determine the success of this approach, we must analyze the level of security as well as the implementation complexity for any possible algorithms that combine transpositions, substitutions and data padding.

We define two metrics for security:

1. **S** is the size of the brute-force search space. Given **S**, an application developer may determine if it is sufficiently large enough for their particular application.
2. **M** is the number of plaintext/ciphertext pairs necessary in order to determine the internal state of the security module. The security module sends a new key before **M** messages is sent. Of course, key transmission overhead must not dominate. **M** must be large enough that a sufficient amount of communication may occur before a new key must be securely transmitted.

In addition, we define one metric for implementation complexity:

1. **C** is the complexity of the algorithm, normalized to some base operations: basic compute, memory load and store operations, as well as a basic random number generation operation. The symbols used to represent each of these operations in equations will be: **op**, **ld**, **st** and **rand**, respectively.

We expect all possible algorithms to have a tradeoff between security and complexity. The more secure, the more complex; and thus, the slower the performance. The key is to provide a precise analytical model of security and complexity so that an application developer may determine if the approach is suitable for their application and deployment environment.

4. DESIGN

In this section, we describe how three basic security techniques (substitution, transposition and data padding) may be applied in the RPC marshalling layer. These operations are used to make all of the RPC messages look the same in terms of their structure, so that any particular RPC message could be invoking any of the methods on the destination server.

4.1 Substitution

Substitution replaces each character in the plaintext with a different character in the ciphertext. Conceptually, substitution is implemented with substitution tables, which enables the individuals with access to the tables to encode/decode messages one character at a time. The substitution table is the “secret” which must be kept from adversaries. Historically, static substitution tables are used to determine the mapping, which means that the table does not change for some period of time. A major drawback of static substitution tables is that if an adversary obtains the plaintext and ciphertext of a message, he can easily reconstruct the table, making it possible to immediately decode all future messages. Another drawback is that they are susceptible to frequency analysis attacks, where the frequency of characters in the plaintext and ciphertext can be used to determine the substitution table.

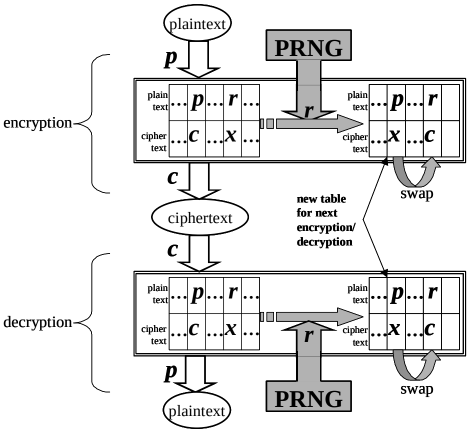
Since we anticipate an adversary being able to eventually decode RPC messages, it is inadvisable to use static substitution tables, as an adversary would be able to reconstruct the table over time. Instead, we use dynamic substitution tables [12]. Figure 1 shows how the table entries are altered every time they are used. Dynamic substitution tables not only prevent table reconstruction, but they also avoid frequency analysis attacks.

Table 1: Security and performance metrics for sample algorithms, assuming reasonable use of registers to reduce memory load/store operations [4]. n **is the number of items in the message,** k **is the number of bits used to hold the method offset ,** b **is the number of random bits per random number used in SHUFFLE algorithm and** p **is the number of padding arguments**

	S	M	C
Method offset substitution	$2^{\mathbf{k}}$	$2^{\mathbf{k}}+1$	Send: 4 **op** + 2 **ld** + 4 **st** + 1 **rand** Recv: 7 **op**+ 3 **ld** + 7 **st** + 1 **rand**
SHUFFLE	**n**!	$\mathbf{bn}\log_{\mathbf{n}!}2$	11**n op** + 2**n ld** + 2**n st** + **n rand**
Padding	(**n**+**p**)!/**p**!	$\mathbf{b}(\mathbf{n}+\mathbf{p})\log_{(\mathbf{n}+\mathbf{p})!}2$	11**p op** + 2**p ld** + 2**p st** + 2**p rand**

Dynamic substitution requires output from a pseudo random number generator (PRNG) every time the table is used. Depending on the performance of the PRNG, this could make applying substitutions to every piece of data in the message quite expensive. One piece of data that must be substituted, however, is the method identifier. For this discussion, let us assume that the method identifier is an offset into an array, as is the case in Java's RMI (Remote Method Invocation) layer. If the method identifier is not substituted, but simply placed into a different location in the message using transposition, then it will be fairly trivial for an attacker to determine the remote method being invoked[1]. Thus, while we may want to examine applying substitutions on all data in the message, it is absolutely necessary to apply a substitution on the method identifier.

Figure 1: Dynamic Substitution



An additional benefit to substituting the method identifier is that probes of the network can be detected. Specifically, the range of numbers to which the method offsets are mapped should be significantly larger than the actual number of methods in the interface. Thus, if an adversary attempts to probe the network with some random values just to see what happens, it is likely that the probe message will contain a method offset value which does not map to an actual method. For the sample component described later in this section, almost 84% of all possible method offsets do not point to a real method. Once a probe is detected, the security system may notify an intrusion detection system and take evasive actions.

Table 1 gives the values of the security and performance metrics. The search space, **S**, equals $2^{\mathbf{k}}$, where **k** is the number of bits used to encode the method offset. Assuming the PRNG is good (i.e. it does not get into a short cycle), the minimum number of messages to determine the random numbers used, **M**, is $2^{\mathbf{k}}+1$. The implementation complexity, **C**, equals 4 **op** + 2 **ld** + 4 **st** + 1 **rand** on the send side, and 7 **op** + 3 **ld** + 7 **st** + 1 **rand** on the receive side. As an optimization, the random number may be generated in advance.

Table 2 gives the values of the metrics for a sample component which has 41 methods but uses 8 bits to encode the method offset using dynamic substitution. This results in **S** = 256 and **M** = 257. While the search space is not large for this particular technique, the number of messages before an intruder may predict the internal state is more than sufficient. For example, our implementation's secure key-exchange takes on order of 108 milliseconds, but provides enough bits of entropy to reseed the PRNG 16 times. Thus, a key must be exchanged every 4096 messages, resulting in a 15% overhead if the component is communication bound.

4.2 Transposition

Transposition does not change the values of the data being sent, but changes the order in which they appear in the message. A particular order is called a permutation.

Transposition can be applied in the RPC layer simply by changing the order in which data is marshaled onto the wire. In order to disperse complex data structures throughout the message, the order should be changed on the primitive data (i.e. bytes, or words). Once an order is decided, it costs very little to alter the marshaling calls to adhere to that order. Indeed, the most time consuming aspect of transpositions at this level is determining the desired permutation of the message.

There are a variety of algorithms in the literature which could be used to determine a permutation based on random numbers [5, 8, 10, 11, 13, 14]. It is necessary to analyze any possible algorithm in terms of the security and complexity metrics introduced in Section 3.1. To give an example of what is feasible, we briefly describe an algorithm based on the SHUFFLE algorithm [5, 8].

[1] There are two reasons that permutation-only of the method identifier results in a trivial attack. First, many of the data values won't fit into the range of method identifiers, allowing an intruder to immediately eliminate them as possible method identifiers. Second, the number of data values that could possibly represent method identifiers will be dramatically less than all possible method identifiers, substantially reducing the search space (and thus search time).

Table 2: . This table lists the values of the security and performance metrics for a sample component which has 41 methods, with n **= 20 ,** k **= 8,** b **= 64 and** p **= 10 .**

	S	M	C
Method offset substitution	256	257	Send: 4 **op** + 2 **ld** + 4 **st** + 1 **rand** Recv: 7 **op**+ 3 **ld** + 7 **st** + 1 **rand**
SHUFFLE	2.43 x 10^{18}	21	220 **op** + 40 **ld** + 40 **st** + 20 **rand**
Padding	7.30 x 10^{25}	18	110 **op** + 20 **ld** + 20 **st** + 20 **rand**

In the SHUFFLE algorithm, an array of data is manipulated, resulting in a permutation of the original array. In our modified algorithm, we shuffle an array of *positions* (1 through **n**, where **n** is the number of data items), and use the position permutation to drive the data marshaling order. This allows us to determine the permutation before the data is available. Thus, if we incorporate data padding as described in the next subsection to make all of the messages the same length, the permutation algorithm may be computed in advance during CPU idle time, reducing the message latency experienced by the application.

```
//param n: number of items to permute

int [] SHUFFLE(int n){
   float u;
   int k, current, tmp;

   int *items = malloc(n * sizeof(int));

   //initialize array of positions
   for(k=0; k < n; k++)
      items[k] = k;

   for(current=n-1; current > 1; current--){
      //generate random number between 0 & 1
      u = random(0,1);

      // make into int between 1 & current
      k = floor(current*u) + 1;

      // swap items[current] and items[k]
      tmp = items[k];
      items[k] = items[current];
      items[current] = tmp;
   }
   return items;
}
```

Figure 2: modified SHUFFLE algorithm

As the pseudocode shows in Figure 2, our modified SHUFFLE algorithm starts with an array the size of the number of items to be permuted, with each entry in the array initialized to its index in the array. Then, we set the current position to be at the end of the array. We randomly choose an index in the array between the beginning and the current position. Swap the value at the randomly chosen index with the value in the current position, then decrement the current position. Repeat until the current position is at the beginning of the array. Now, the value at index **x** in the array is the position in the message for the data normally sent in position **x**.

Table 1 shows the equations for the security and complexity metrics of our modified SHUFFLE algorithm. There are **n**! possible permutations of the message, where **n** is the number of items to be permuted in the message. Table 2 shows that for our sample component with the number of bits per random number, **b** = 64 and the number of data items, **n** = 20, **S** = 2.43 x 10^{18} (or approximately 2^{62}). On average, an adversary would have to be able to analyze 2^{61} states to find the actual state. If an attacker had a cluster of 1 GHz machines available to her and if each machine could analyze a state in 20 cycles, she would require over 45 billion nodes to decode the message in 1 second, or approximately 12.5 million nodes to determine the message in 1 hour.

To compute **M**, we determine how many sequences of random numbers could have resulted in a particular permutation. Then we can determine how many messages are needed to eliminate all but one sequence. When **b** random bits are used in each iteration of the loop, **M** is equal to $\mathbf{bn}\log_{\mathbf{n}!}2$. In Table 2, we see that **M** is 21 messages for the sample component. Using the key exchange and message latencies that we used in Subsection 4.1, this would result in a key exchange every 320 messages with an overhead of 69%. While this may appear large at first, we believe the overhead can be reduced by performing parts of the key exchange in the background before the key is needed.

Finally, we compute the complexity of the algorithm, assuming that the compiler can make judicial use of registers, avoiding memory load/store operations for temporary data like temporary variables and loop iterators. The complexity then becomes 11**n op** + 2**n ld** + 2**n st** + **n rand**.

4.3 Data Padding

Data padding consists of adding data to a message. It is often used to ensure messages are a particular length, making the implementation of certain algorithms on the message easier. In addition, data padding has been used to avoid traffic analysis attacks, which are able to infer important information simply by knowing how much data is being sent [15].

For RPC communication, data padding can be used to avoid traffic analysis attacks which can determine the identity of the remote method simply by analyzing the length of the message and the values of the arguments being passed to the method. Adding padding data makes all of the messages look identical in terms of their length and the type of data that is being sent. Thus, to an eavesdropper, any message may be used to invoke any of the methods on the server.

Data padding can be applied to the message in the marshaling layer. For each piece of padding data, two things must be decided: its location in the message and its value. There are multiple approaches to determine both. To identify the most suitable, the security and performance metrics must be applied. Here, we have space to describe one possible approach.

To determine the location of the padding data, we first append it to the original message. It then undergoes transposition along with the real data as described in the previous section. To determine the value, we could simply send a random number from the PRNG. More intelligent value choices can be made depending on the types of the real data in the message, making the values of the data statistically meaningless to an eavesdropper. For example, if a Boolean is sent, it would be wise to send the negation of that Boolean value as well. Then, an eavesdropper cannot easily determine the value of the Boolean.

Table 1 provides equations for our metrics based on sending random numbers as our padding data. It does not make sense to insert padding data without also permuting the message data, so the security metrics, $\mathbf{S}$ and $\mathbf{M}$, contain equations which include the transposition algorithm. Thus, $\mathbf{S}$ is $(\mathbf{n}+\mathbf{p})!/\mathbf{p}!$, where $\mathbf{p}$ is the number of pieces of padding data in the message. Similarly, $\mathbf{M}$ is $\mathbf{b}(\mathbf{n}+\mathbf{p})\log_{(\mathbf{n}+\mathbf{p})!}2$. The equation for $\mathbf{M}$ assumes that the values of the padding data are checked on the receiver side to ensure that they are accurate. This check aids in detecting probes of the network in a way similar to the one described in Subsection 4.1. The complexity of this algorithm is the complexity of the transposition algorithm (replacing $\mathbf{n}$ with $\mathbf{p}$) plus one random number for each piece of padding, for determining the value of the pad: $11\mathbf{p}\ \mathbf{op} + 2\mathbf{p}\ \mathbf{ld} + 2\mathbf{p}\ \mathbf{st} + 2\mathbf{p}\ \mathbf{rand}$. Of course, the complexity metric does not include the time it takes to send the padding data ($\mathbf{p}$ times the data rate), but that should be considered as a cost.

As an optimization, all of the overheads associated with the permutation algorithm may be pre-computed. The data padding values, however, can only be determined once the identity of the remote method being invoked is known, as the number and type of pads is dependant on the remote method.

5. INITIAL EVALUATION

We have implemented a proof-of-concept prototype to demonstrate the feasibility of this approach. The prototype performs a diffie-hellman key exchange at connection setup time. Based on the key, it performs a simple method offset substitution, message permutation and data padding algorithm in the RPC layer. The purpose of the prototype is to prove the concept of implementing this scheme in the RPC layer, and therefore, it does not explore the many possible algorithms that could be used to apply the operations to a given method invocation.

Our prototype builds upon a high-performance Java implementation called Manta [17]. Our executing environment is an 8-node network of 1.5 GHz Pentium 4 boxes with 256 MB of RAM, running RedHat Linux 7.1. The cluster is connected with 100 Mb switched Ethernet.

Adding simple substitutions and transpositions to the RPC layer incurs an overhead of less than 10% of the original message latency, showing that this is a promising approach in terms of the necessary performance. For example, for an RPC with 64 data bytes, our mechanism adds 12 microseconds to the base plain-text latency of 152 microseconds.

In addition, since many of the algorithms that we are exploring require one or more random numbers, we have measured the time it takes to retrieve a random number from an implementation of Yarrow. One call to Yarrow took approximately 0.241 microseconds to retrieve 64 random bits. Thus, for our sample component used in Table 2, the time it would take to obtain all of the necessary random bits is less than 10 microseconds, which is substantially less than the maximum overhead requirement of 100 microseconds. It may be possible to further reduce that overhead by eliminating function calls and retrieving all of the random bits in one call.

We believe that further optimizations can be performed to effectively reduce the latency experienced by the application. Because much of the work in the algorithms we have presented does not depend on the actual data being sent, several key pieces may be computed in advance during CPU idle time. Possibilities include:

- random number generation
- computation of the message permutation
- diffie-hellman key exchange

6. RELATED WORK

Globus [7] is one of the only high-performance projects which has added an encryption option (DES) to their standard communication library. Because of the cost, it is not clear if this capability has been adopted by their users for communication within a high-performance cluster.

The Globus [7] and Legion [6] architectures contain mechanisms for users to specify the level of security required by each communication channel. While neither project provides a low-overhead/short-cover-time mechanism as we have described in this paper, both could include such a mechanism in the future. With the variety of communication patterns in high-performance environments, we believe the flexibility provided by these two architectures is essential for satisfying the necessary security and performance requirements.

A seminal work on Lightweight Remote Procedure Calls optimizes RPCs between processes on the same machine [2]. This concept could be incorporated in any high-performance messaging layer, disabling all security/encryption of the data when being sent to another processes on the same machine. The focus of our paper, however, is on RPCs that traverse the network.

7. CONCLUSIONS

Because of the performance requirements in high-performance distributed systems, it is not possible to simply retrofit existing security mechanisms and expect the HPC community to use them.

This paper is a first attempt to construct a security solution based on the specific needs of high-performance component communication. We classified the data security needs and determined that much of the data transmitted over the network has a short cover-time requirement. We then presented an approach which capitalizes on the marshaling infrastructure in order to maintain a low overhead. We specified metrics for evaluating the approach, and analyzed three security techniques with these metrics. Finally, we described an initial prototype and its performance which indicates that this approach is promising for meeting the performance requirements of the high-performance domain.

8. REFERENCES

[1] Allcock, Bill et. al. "Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing". *IEEE Mass Storage Conference*, 2001.

[2] Bershad, Brian et. al. "Lightweight Remote Procedure Call". *ACM Transactions on Computer Systems*, Vol. 8, issue 1, pages 37-55, February 1990.

[3] Clar, David D., Van Jacobson, John Romkey, and Howard Salwen. "An analysis of TCP processing overhead". *IEEE Communications Magazine*, June 1989.

[4] Connelly, K. and A. Chien. "Elusive Interface Design and Analysis", Computer Science Department, Indiana University. April, 2002.

[5] Durstenfeld, Richard. "Algorithm 235: Random Permutation [G6]". *Communications of the ACM*. Vol. 7, page 420, 1964.

[6] Ferrari, Adam et. al. "A Flexible Security System for Metacomputing Environments". *High Performance Computing and Networking Europe*, April 1999.

[7] Foster, Ian, Nicholas Karonis, Carl Kesselman and Steven Tuecke. "Managing Security in High-Performance Distributed Computations", *Cluster Computing*, Vol 1, issue 1, pages 95-107, 1998.

[8] Knuth, Donald. 1997. *The Art of Computer Programming*, Vol 2, *Seminumerical Algorithms*. 3rd ed. Reading, Mass: Addison-Wesley.

[9] Kay, J. and J. Pasquale. "Measurement, Analysis and Improvement of UDP/IP Throughput for the DECstation 5000", *Proceedings of the 1993 Winter Usenix Conference,* San Diego, USA, pages 249-258.

[10] Plackett, R. "Random Permutations". *Journal of the Royal Statistical Society, Series B (Methodological)*. Vol. 30, issue 3, pages 517-534, 1968.

[11] Rao, C. "Generation of Random Permutations of Given Number of Elements Using Random Sampling Numbers". *Sankhya, A*. Vol. 23, pages 305-307, 1961.

[12] Ritter, Terry. "Substitution Cipher with Pseudo-Random Shuffling: The Dynamic Substitution Combiner". *Cryptologia* Vol. 15, issue 4, pages 289-303, 1990.

[13] Sandelius, Martin. "A Simple Randomization Procedure". *Journal of the Royal Statistical Society: Series B (Methodological)*. Vol. 24, issue 2, Pages 472-481, 1962.

[14] Sloane, N. "Encrypting by Random Rotations". *Cryptography: EUROCRYPT'82*. Lecture Notes in Computer Science Vol. 149, pages 71-128, 1983.

[15] Timmerman, Brenda. "A Security Model for Dynamic Adaptive Traffic Masking". *New Security Paradigms Workshop*, 1997.

[16] Tuecke, S. et. al. "Grid Service Specification". February 2002. http://www.globus.org/research/papers/gsspec.pdf

[17] Veldema, Ronald, Rob van Nieuwpoort, Jason Maassen, Henri E. Bal and Aske Plaat. "Efficient Remote Method Invocation". *Technical Report IR-450*, Vrije Universiteit Amsterdam, September, 1998.

Dynamic Resource Discovery for Applications Survivability in Distributed Real-Time Systems

Byung Kyu Choi
Department of Computer Science
Michigan Technological University
Houghton, MI 49931-1295, USA
bkchoi@mtu.edu, Tel:1-906-487-3472
Fax:1-906-487-2283

Sangig Rho Riccardo Bettati
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112, USA
{sangigr, bettati}@cs.tamu.edu
Tel:1-979-845-5469, Fax:1-979-847-8578

Abstract

In this paper we propose a new resource discovery protocol, REALTOR, which is based on a combination of pull-based and push-based resource information dissemination. REALTOR has been designed for real-time component-based distributed applications in very dynamic or adverse environments. REALTOR supports survivability and information assurance by allowing the migration of components to safe locations under emergencies like external attack, malfunction, or lack of resources. Simulation studies show that under normal and heavy load conditions REALTOR remains very effective in finding available resources with a reasonably low communication overhead. REALTOR 1) effectively locates resources under highly dynamic conditions, 2) has an overhead that is system-size independent, and 3) works well in highly adverse environments. We evaluate the effectiveness of a REALTOR implementation as part of Agile Objects, an infrastructure for real-time capable, highly mobile Java components. [1]

1 Introduction

Traditionally, the main purposes of distributed systems has been the sharing of expensive computing resources or the clustering of large numbers of cheap resources so that overall performance of computing-oriented large-scale applications can be significantly improved. As security issues have become relevant, application survivability and information assurance must be addressed in high performance distributed computing as well [7, 8]. *Application survivability* requires that applications are dynamically reconfigurable during run-time, especially when the system is under external attack. For component-based systems this means that components may want to migrate to locations that are not being attacked or to locations that run at higher security levels. Similarly, component migration increases the level of information assurance, as critical data in the system can be kept safe from localized external attacks. In the type of applications described above, both resource availability and resource requirements can fluctuate widely: As nodes in the system come under attack, resources on these systems become unavailable. At the same time, components on these nodes migrate, and so change the resource availability across the system.

We recognize that *resource discovery* is of prime concern because the rest of the migration procedure depends of the performance of resource discovery and allocation. In general, it is a matter of resource information dissemination with two key parameters. One is *when* to advertise resource information, and the other is to *whom* the information should be sent. In other words, this dissemination is controlled by advertisement *interval* and the *scope* of neighbors that are intended to receive new advertisement.

In order to effectively support migration, the resource discovery should meet the following requirements: *scalability*, *effectiveness*, and *attack-survivability*. By scalability we mean that the overhead of resource discovery should be independent of the network size or the system size in terms of numbers of nodes, resources of application components. By effectiveness we mean that the resource discovery should be able to find a host which is able to accommodate the migration request when such a host exists in the system. By attack-survivability we mean that the resource discovery should not be affected by external attacks.

In this paper, we propose REALTOR (REsource ALlo-

[1] This work was partially sponsored by DARPA under contract number F30602-99-1-0534 and by the Texas Higher Education Coordinating Board under its Advanced Technology Program.

caTOR) as a solution for resource discovery for highly dynamic environments and applications with timing guarantees. REALTOR satisfies the requirements described above by having: 1) a very light communication protocol that is scalable independently of the network size, 2) a dynamic neighborhood concept that is independent of the physical distance (for example, hop count), and 3) a stateless protocol which makes the system idempotent and so inherently fault tolerant.

The rest of this paper is organized as follows. In Section 2, we briefly discuss the related work. We describe REALTOR in detail in Section 3. In Section 4, we describe the Agile Objects project, on which we base REALTOR. A comparison study of REALTOR is provided in Section 5 with simulation results. Real measurements using the Agile Objects infrastructure on a 20 node Unix workstation cluster is provided in Section 6. Finally conclusions and future work are in Section 7.

2 Related Work

In this section, we survey representative research on resource management for distributed computing systems, for example, Globus [10] and Legion [12, 17] for multi domain scale, and Condor [15] for dedicated or clustered environments. The authors in [16] presents a taxonomy that provides a good insight into the overhead of information dissemination for resource discovery. According to this, resource discovery can be described as follows: 1) centralized vs. decentralized organization 2) push vs. pull information dissemination, 3) aperiodic vs. periodic dissemination interval. We use this taxonomy in this paper as it appropriately classifies resource discovery methodologies.

Globus encompasses many research issues under the name of "virtual organization", which is primarily a coordinated large-scale dynamic resource sharing and problem solving system over multi-institutions. Globus has developed its own resource management architecture, GARA (Globus Architecture for Reservation and Allocation) [9]. Unlike per-session on-demand resource reservation (RSVP [20], for example) GARA focuses on advance reservations and co-allocation with which it can easily enhance end-to-end QoS [11]. In this project, a resource discovery based on the peer-to-peer model has been proposed [14], which consists of a few request-forwarding algorithms in a fully decentralized architecture accommodating heterogeneity and dynamism in resource.

Legion provides another distributed computing infrastructure in very large-scale systems. In its resource management [6], however, the prime interest was in supporting and matching user task requirements and the autonomy of local domain. Interestingly, it provides both push-based and pull-based resource discovery.

Condor [15] provides resource management services that harness the capacity of very large collections of distributively owned UNIX workstations. The need for maximum computation throughput has been the driving force for the efficient utilization of distributed computational resources [3], and a metric, "Goodput" has been proposed for co-scheduling CPU and network capacity [2]. For resource discovery, a framework "Matchingmaking" has been proposed [18], which separates matching and claiming phases of resource allocation.

Although our survey is not exhaustive, we still observe the followings. Work on resource discovery in large-scale distributed systems has initially focused on the functionality of the resource discovery protocol and on appropriate visualization [19], with emphasis on protocol specifications and resource representations. This has been followed by some performance issues, including scalability and adaptability [5] as response to highly dynamic system conditions. For example, scalability has been studied in terms of message overhead for information dissemination approach for reducing the overhead of information dissemination is described in [16].

None of this work has addressed the *effectiveness* of resource discovery and allocation, a measure that is of particular importance in distributed real-time systems. By effectiveness we mean the ability of the resource discovery system to find and allocate available resources in overload situations. We note that overload situations are particularly problematic for QoS sensitive applications, which do not degrade gracefully with decreasing amount of available resources. Interestingly however, many papers addressing resource discovery do not consider effectiveness when evaluating their proposals. In this paper, we propose a scalable and effective dynamic resource discovery based on a combination of *pull*-based and *push*-based methods. We then evaluate our proposal by simulation experiments and measure its performance in an implementation.

3 Agile Object Systems

We realized REALTOR as part of the UCSD/Texas A&M Agile Objects project [1]. Agile Objects provides an infrastructure for dynamically re-configurable distributed component-based applications. Components in such applications are capable of migrating frequently, which provides them with *location elusiveness*. The latter is greatly beneficial for both survivability, as the application is able to quickly reconfigure during attacks, and for information assurance, as the location and tracking of critical components become significantly more difficult for an attacker. Figure 1 shows a high-level diagram of the Agile Object System. The main objective of *REALTOR* is to provide pro-active resource management for fast migration. This is achieved by keeping track of resource availability in neighbor hosts so that migration can be processed immediately when needed, without need for expensive signaling during migration. *Admission Control* is in charge of the admission decision, component instantiation, and migration. REALTOR's objective is to maintain a list of hosts with their resource status, so the admission control can be very light-weight and happens concurrently to the migration properly. In those rare occurrences where REALTOR directs a migration to an overloaded node, migration is aborted and the next node in REALTOR's list is tried. The management of CPU resource is greatly simplified by the use of guaranteed-rate scheduling in the nodes. This greatly reduces the admission control overhead, which becomes a simple utilization test, and available CPU resource can be directly measured in terms of unallocated utilization. The current implementation uses a Constant Utilization Server [4]. The mechanics of component migration is handled by the migration subsystem.

During migration, the component state is moved, if necessary code and libraries at the destination are updated and service access points are transferred. In addition, the naming service is updated to reflect the new location of the component. The correct realization of Agile Objects is based on an extension of Java RMI. The agile software components are realized as migratable objects in Java RTSJ, with the Constant Utilization Servers realized as extensions of RTSJ schedulers. The guaranteed-rate scheduling at the nodes allows for an accurate definition of resource requirements during design and deployment time, and thus eliminating the need for cumbersome resource reallocation mechanisms during run-time and for priority inheritance extensions to RMI.

Migrations can be either *application triggered* or *resource triggered*. In the former case, either the application itself, or third-party applications (security enforcers, for example) triggers the migration. In the latter case, migration can be triggered by schedulers and resource monitors as response to overload. When a migration is triggered in either way, the request is forwarded to the REALTOR component at the node (REALTOR A in Figure 1). REALTOR A then returns a list of hosts to Admission Control A, where the necessary resources are currently available for newly migrating objects (3). The list is updated by the REALTORs according to the unavailability of local resources (2). The detailed updating procedure is described in the next section. Receiving the list, Admission Control A begins negotiation with the admission controls in the list (6). If one of the host admits the migration request, then Admission Control A asks Migration Module A to actually move the object (7). At the same time Admission Control A informs Job Scheduler A of this migration (5). Migration Module A now moves the object to Host B (9). Upon receiving the object, Migration Module B registers the object with Job Scheduler B (8). The migration of the component can happen concurrently to the negotiation among the Admission Controls (speculative migration), thus enabling very low-latency migration.

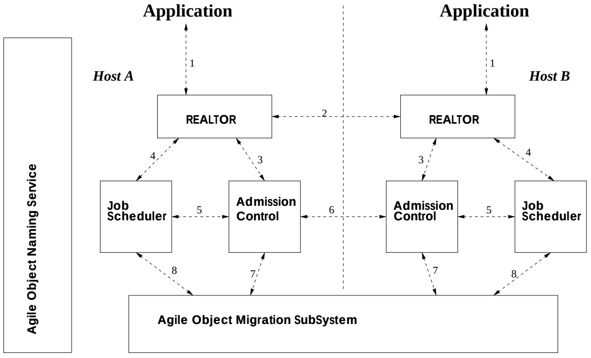


Figure 1. Software Components of Agile Object System

4 REALTOR: REsource ALlocaTOR

Given the scale and the volatile nature of the agile systems considered in this work, the resource discovery and allocation system must satisfy a number of requirements: First, the resource availability information must be readily available at any time so that any host under attack or malfunction is able to locate a host and move the software components immediately. Any resource allocation scheme must be *pro-active*, as nodes are in need for migration. Second, any resource discovery scheme for this type of systems considered here must be largely *state-less*. Nodes leave and join the system at any time, due to attacks and failures, or after recovery. In REALTOR we rely heavily on *soft state*, which is be re-freshed at low cost in order to retain an accurate view of resource availability in the system. Third, the protocols must be largely *idempotent*, so that node failures do not give raise to errors. Finally, given the large amount of dynamics in the system and the need to support scalability without loss of information accuracy, the resource discovery mechanisms at any node should interact only with a *small subset of other nodes*. We use the concept of *community* in REALTOR, which links a potential resource user with a community of potential resource providers. Communities are ephemeral in nature: they spontaneously appear, change over time, based on resource requirements, resource availability at the nodes in the community, and the status of nodes in the community.

REALTOR: Each host establishes its own community for future software component migration, which is a set of nodes able to receive a migrating component. Each host is free to join as many communities as it is able to without over-allocating its spare resources. Therefore, each host usually owns one community and is a member of several other communities. The membership in a community is not static, and must be refreshed. The membership of a node in a community is valid only for the interval between two consecutive refresh messages. So, in order to maintain the membership to a community, a host needs to keep responding to all refresh messages from the organizer. When a member stops responding to refresh messages from the organizer, it de facto leaves the community. Similarly, when a community organizer stops sending refresh messages, the community will naturally disband. Communities are established and managed with the community protocol described below.

Community protocol: The community protocol was designed with three immediate goals, 1) the protocol should be *effective* in finding available resources within its own community, 2) the protocol overhead should be independent of network size, and 3) the protocol should be *stateless*. Therefore, Community protocol has only two types of messages.

HELP: When a host joins the system, it begins to build its own community for software component migration in the future. The invitation to the new community is done by broadcasting a HELP message to the network[2]. The interval between two consecutive HELP messages is determined by *Algorithm H*, which we describe below.

PLEDGE: When a host receives a HELP message, it determines whether to join or not the community. Once it determined to do so, it sends a PLEDGE message to the community organizer (i.e., the originator of the HELP message) whenever its resource usage status changes across a threshold level. The threshold level is determined by *Algorithm P* at each local host.

The message formats are defined as follows:

HELP: Hostid (community organizer identifier), Type(help), The number of current members (number of members), The urgency of the resource request (degree of demand).

PLEDGE: Hostid (identifier of the pledger), Type(pledge), Resource availability (degree), Number of communities of which it is a member (number of communities), Probabilities of resource grant when requested (distribution).

Algorithm H: As can be seen in Figure 2, a host keeps sending HELP messages at the rate of one message every HELP_interval time units as long as a task arrives and its resource usage is above a threshold. The length of HELP_interval changes over time depending on the success of finding available resource. If it succeeds, HELP_interval is decreased by the proportional amount of *beta* as a reward, while it increases the interval by the proportional amount of *alpha* as a penalty. By using both reward and penalty, the interval shrinks if there are resources available and expands if there are not. The idea is to avoid unnecessary discovery activity of resource when the whole system is heavily loaded. Upper_limit prevents an unbounded increase of HELP_interval after a series of failure in finding available resources. The speed of expansion or shrinkage is controlled by appropriately setting alpha and beta values.

```
Algorithm  H
Input: Time_current, Time_sent
Output: HELP message

Whenever a task arrives   do {
        If resource usage would exceed a threshold level  {
                 If ((T_current − T_sent) > HELP_interval) {
                          send HELP ;
                          set_timer;
                 }
        }
}


Timeout do {
        If ((HELP_interval + HELP_interval * alpha) < Upper_limit)
                 HELP_interval +=  HELP_interval * alpha;
}

Whenever a PLEDGE message arrives do {
         If the corresponding timer is not expired
                 reset_timer;
        Update corresponding PLEDGE list;
        If a node is found for migration {
                 If ((HELP_interval − HELP_interval * beta) > 0)
                         HELP_interval −= HELP_interval * beta;

        }
}
```

Figure 2. Algorithm H in REALTOR

Algorithm P: As can be seen in Figure 3, the host replies with a PLEDGE as long as 1) a HELP message arrives and 2) its resource usage is below the threshold level. Also, once a host determines to be a member of a community, it replies with PLEDGE messages whenever its resource usage status changes across the threshold level. This helps the organizer keep the most current information. The value of alpha and

```
Algorithm P
Input: HELP messahe
Output: PLEDGE message

Whenever a HELP message arrives do {
         If the host has used its resource less than a threshold level
                     Reply PLEDGE;
}

Whenever the resource availability changes across the threshold level do {
          Reply PLEDGE;
}
```

Figure 3. Algorithm P in REALTOR

[2] Networks in this context are typically application-level overlay networks.

beta (in Figure 2) are subject to the local resource manager.

Pure PUSH: Each host disseminates its own resource availability information to its neighbors unconditionally at every preset interval. In comparison to REALTOR, there is only periodic PLEDGE message without HELP.

Pure PULL: Each host solicits PLEDGE from its community members whenever 1) a task arrives and 2) the resource usage level is beyond a threshold level. In comparison to REALTOR, this scheme generates HELP messages unlimitedly (without Upper_limit in Algorithm H) as long as resource usage is above the threshold level.

Adaptive PUSH: Each host disseminates its own resource availability information to its neighbors whenever the resource usage changes across a threshold level. In comparison to REALTOR, PLEDGE is automatically generated at each major status change without solicitation (HELP).

Adaptive PULL: Each host solicits PLEDGE from its community members whenever 1) a task arrives, 2) the resource usage level is beyonds a threshold level, and 3) a time window has passed since the previous HELP. In comparison to REALTOR, this scheme generates HELP messages in the same fashion as in REALTOR. It is different from REALTOR, however, in that it generates PLEDGE exactly once in response to each HELP.

5 Experimental Performance Evaluation

Here, we compare the performance of REALTOR with those of the alternative resource discovery protocols introduced previously in this paper, using a set of simulation experiments. Unlike REALTOR, which advertises resource status when solicited using HELP and PLEDGE (PUSH and PULL-based), other protocols disseminate resource status information to the neighbors periodically, with or without solicitation from other nodes, using either PUSH-based or PULL-based schemes only. We, therefore, measure the performance in terms of *message overhead* and *effectiveness* in finding available resources. Since the communication pattern in the purely PUSH-based approach is independent of the available resources and their location, this approach tends to either waste communication bandwidth or fails to appropriately locate resources. A purely PULL-based approach may suffer from high volume of HELP messages under overloaded conditions because most hosts cannot pledge.

For the experiments, we simulate the mesh topology displayed in Figure 4, with 25 nodes and 40 links. Each intersection represents a node. For fair comparison purposes, we assume that the topology represents the limited scope of neighbors for REALTOR and all other four resource discovery schemes. In reality, there should be a mechanism in place limiting the scope of neighbors for REALTOR, for example, as an IP multicast group. We compare the communication overhead and the effectiveness of the five approaches by simulating their behavior under increasing load in the nodes. For this, we randomly generate task at increasing rates, and assign them randomly to a node. The resource discovery and allocation algorithms then must migrate the tasks, when needed, to nodes with available CPU capacity.[3]

We generate tasks with exponentially distributed lengths of a mean value μ. The generated task is given to a node randomly selected from Node 0 through Node 24. The task arrival forms a Poisson process with a rate of λ. Each node is assumed to have a single queue of 100 seconds to process tasks. Task lengths are defined in seconds with a mean value of 5. So, a task with value 2 holds the CPU on the node for 2 seconds. Tasks arriving at a node whose queue is already full are supposed to migrate to another node whose queue can still accommodate the task. In these experiments, in order to satisfy the requirement of *pro-activeness* for immediate migration, we measure the performances of the five approaches with only a one-time migration try to the best candidate destination node provided by each approach. So, if the candidate destination node cannot accommodate the migrating task, then the task is rejected.

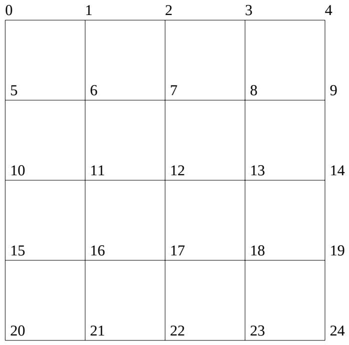


Figure 4. The network topology for the simulation

For this simulation, we use a simple threshold strategy for both *Algorithm H* and *P* in PULL, Adaptive-PULL, Adaptive-PUSH, and and REALTOR cases. *Algorithm H* sends out HELP messages when the queue length exceeds a certain level. The queue level is checked whenever a new task arrives. So, the HELP messages are sent out whenever the three conditions are met: 1) a new task arrives, 2) the queue including the new task exceeds a certain level, and 3) the time window has passed. Likewise, *Algorithm P* replies HELP with PLEDGE whenever the two conditions are met: 1) a HELP message arrives once, and 2) the queue is occupied below a certain preset level. Also for simplicity,

[3] In this simulation, we assume a single resource - CPU. More general resource scenarios such as network bandwidth, current security level, etc., would give similar results.

the number of messages for resource information advertisement to the network is counted as the number of links for all approaches. This assumption does not affect the performance comparison. So, in REALTOR case, HELP message requires the number of links for flooding, while PLEDGE message takes the average number of shortest paths, which is 4 in this particular network topology. So the total number of messages is counted as the sum of 1) message flooding, and 2) communication for migration between admission controls. In the following figures in this section, the curve names stand for the followings. We compare the following five algorithms:
"Pull-.9": a pure PULL approach which uses 0.9 for both Algorithm H and P. "Push-1": a pure PUSH which uses 1 second periodic interval for information dissemination. "Push-.9": an adaptive PUSH which disseminates information only when the resource usage changes across a threshold level. "Pull-100": an adaptive PULL which limits HELP_interval from increasing infinitely, in this case the limiting value is 100 time units (Upper_limit in Figure 2). "REALTOR": combination of "Push-.9" and "Pull-100". Algorithm H 0.9: every new task arriving a queue whose length reaches more than 90% including the new task triggers a set of HELP messages. Algorithm P 0.9: means that every HELP message arriving a node whose queue is occupied less than 90% triggers a PLEDGE message.

Figure 5 compares the task admission probability of the five approaches. The x-axis represents the task arrival rate, and y-axis shows the admission probability. This set of curves are obtained this way. First, we run this simulation for Push-1. After obtaining the curve "Push-1", we repeatedly run the simulation for other approaches with different set of simulation parameters until finally we have a set of curves close enough to "Push-1". So, as seen in the figure, "Push-1" lies in the middle of the curves. "REALTOR" and "Push-.9" show the best performance. However, there is no big difference between the performances for all load conditions. We believe that these curves are close enough to assume that they are more or less than equals, which provides the ground on which we can compare the communication overhead for the same performance.

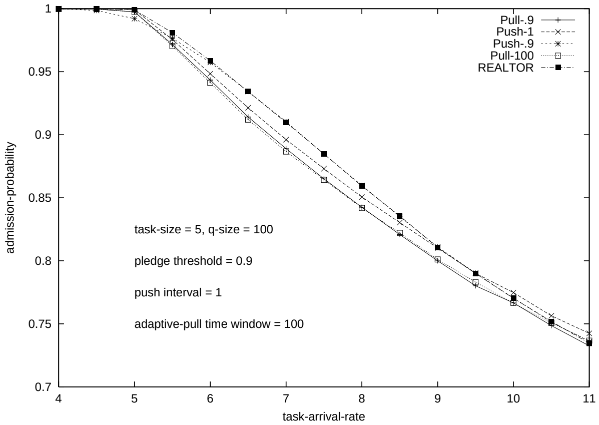


Figure 5. Admission probability

Figure 6 shows the communication overhead of the five approaches. The x-axis represents the task arrival rate, and y-axis shows the total number of message exchanges. As we expected, "Push-1" shows the highest overhead, especially under lightly loaded conditions where it wastes too much communication bandwidth unnecessarily. "Pull-.9" (pure PULL) keeps increasing its overhead as the system gets overloaded. As it increases almost linearly, it will eventually cross "Push-1" for very high arrival rates of λ. However, "Pull-.9" is still below "Push-1" until the admission probability drops to below 0.75, after which the system will be completely overload. It is apparent that the pure PULL approach also wastes much communication bandwidth. On the other hand, "Pull-100" shows the least amount of communication overhead independently from the load. However, this causes poor performance in admission probability in Figure 5. "Push-.9" (adaptive PUSH) shows a moderate overhead and a very close performance to "Push-1" (pure PUSH). Finally REALTOR shows the best performance with still a moderate overhead slightly higher than "Push-.9". In fact, we think that this result is quite expectable because REALTOR combines the two approaches: an adaptive PUSH and an adaptive PULL, so it naturally takes advantages of both while adding a slight amount of communication overhead. The point so far is that REALTOR performs better in terms of effectiveness with a modest communication overhead which is around one third of that of PUSH.

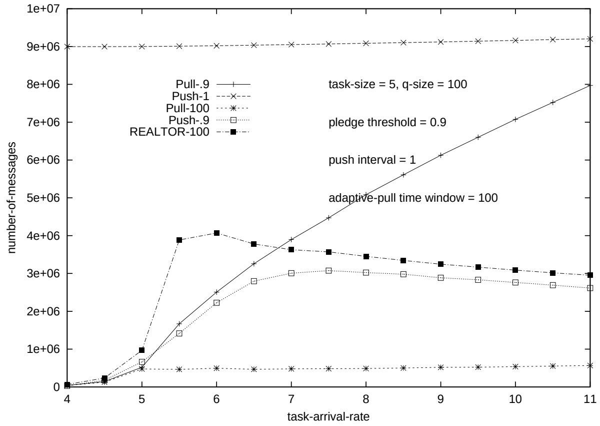


Figure 6. Number of messages exchanged

Figure 7 compares the resource discovery protocol overhead per admitted task. For example, "Push-1" costs 200 message exchanges for a single admitted task at $\lambda = 5$, while all other approaches take about less than 50. The amount of overhead in REALTOR and "Push-.9" decreases as the sys-

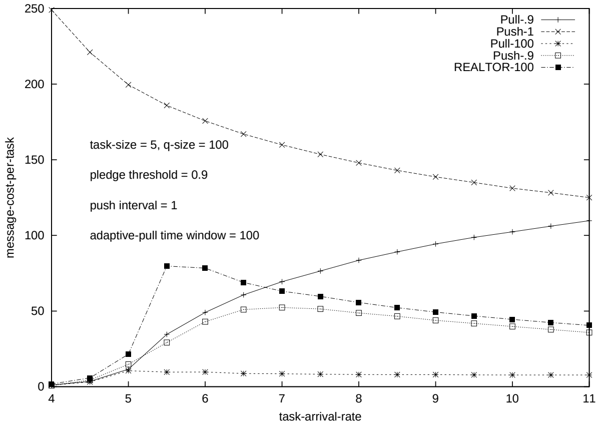


Figure 7. Communication cost per admitted task

tem becomes overloaded. This is because 1) HELP_interval is kept at maximum due to the repeated failure of finding available resources, and 2) since the resource usage level at each host is kept above the threshold level. The reason for the peak around λ = 6 in REALTOR is that the resource usage level changes across the threshold most frequently around that point. The admission probability at that point is about 0.95, which means there are a lot of fluctuations in usage levels, causing PLEDGE messages to be generated. This figure clearly illustrates the cost of disseminating resource information periodically regardless of load conditions. Also, it is very clear that either a pure-PUSH or a pure-PULL do not scale well.

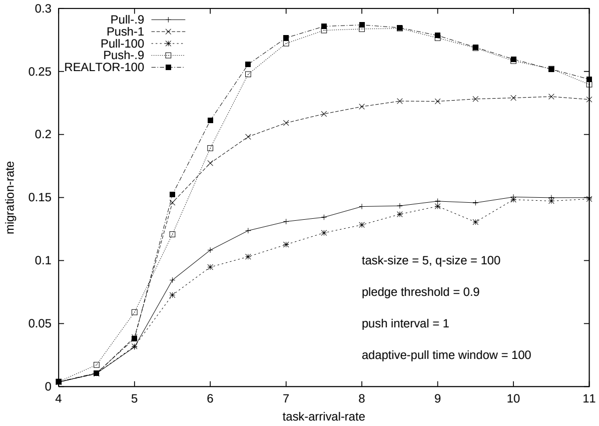


Figure 8. Migration rate

Another way to evaluate the *effectiveness* of resource discovery is to look at the task migration as it is the critical factor in admission probability. Figure 8 shows the migration rate per admitted task for the five approaches. As the previous curves imply, REALTOR has the highest migration rate close to 30% at top around λ = 8, where admission probability is about 0.85. After that the rate slowly decreases due to the suppressed HELP messages by Upper_limit in Algorithm H. In "Push-1" case, the rate keeps increasing rapidly until λ = 7, and very slowly after that as the task arrival rate increases. This is because the periodic dissemination becomes less effective at higher task arrival rates. Both pull-based approaches show the lowest rates, as they show the lowest curves in the admission probability. This is because the *untimeliness* of the pull-based approach. Since, in pull-based approach, information is collected before migration request rises, the information can be out-of-dated rather easily. On the other hand, in adaptive push-based approach, the information is more timely because each host disseminates information only when it changes the status.

We interpret the results as follows. *REALTOR* outperforms either *push-based* or *pull-based* in terms of *overhead* and *effectiveness* regardless of the load conditions. The lessons we learned with this set of experiments are: 1) pure push-based approaches waists resources too much during light-load conditions, 2) pure pull-based also waists resources much in overload conditions, 3) REALTOR performs best by combining both adaptive push-based and adaptive pull-based approaches for any load conditions.

6 Implementation Experience

Here, we describe the measurement results for an implementation of REALTOR within the Agile Object system described in Section 4. Here, the performance is measured in terms of *admission probability*. For the performance measurement, we used a workstation cluster of Linux machines at the Concurrent Systems Architecture Group Laboratory in the University of California San Diego, where Agile Object Project is being integrated. The cluster for this measurement consists of 20 homogeneous hosts running Redhat Linux Version 7.2 Operating System on Pentium II at 450 MHz. Each host is a single server that processes arriving tasks sequentially. REALTOR uses IP multicasting for HELP messages and UDP for PLEDGE messages. Admission Control uses TCP connections for admission negotiation. Job Scheduler provides a simple form of real-time task scheduler with static priority and EDF (Earliest Deadline First) in the same priority. To reduce Java overhead as much as possible, all the components on the same host are implemented in a single JVM (Java Virtual Machine).

The experiment scenario remains the same as in the simulation in the previous section. So, we implement each task as a timer waiting to expire. This considerably simplifies migration, as the only state of the task is the current value of un-expired time. In real situations, the migration time will be longer than that of this experiment depending on the actual size of the software component.

Although we are behind from completing the implementation, Figure 9 shows an earlier measurement of admission probability with 20 hosts. Other parameters remain the same as in the simulation. The curve shows the same type of shape as in the simulation. We are currently measuring the performance of REALTOR in various scenarios.

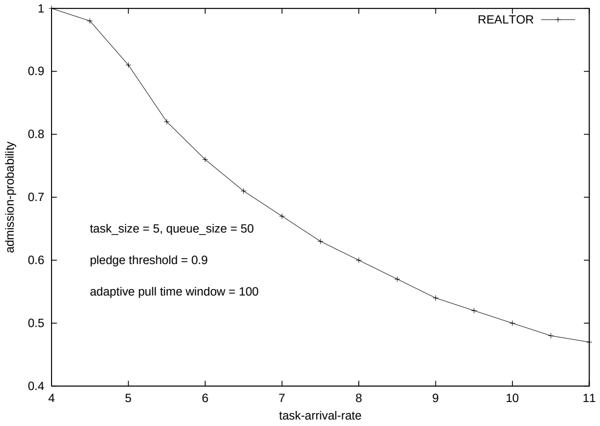


Figure 9. Admission probability measured.

7 Conclusions and Future Work

In this paper we proposed a new resource discovery protocol, REALTOR, based on combination of pull-based and push-based resource information dissemination. REALTOR has been designed for QoS sensitive software application, consists of software components, which support application's survivability and information assurance by migrating to safer places under emergencies like external attack, malfunction, or lack of resources. Simulation studies show that under normal and heavy load conditions REALTOR remains very effective in finding available resources with a reasonably low communication overhead compared to pure push-based approach. According to the simulation results REALTOR 1) effectively finds resources under highly dynamic conditions, 2) has a low overhead that is system-size independent, and 3) works well in highly adverse environments due to its statelessness. Also the performance of effectiveness in a REALTOR implementation is provided with measurements in terms of admission probability in the Agile Objects system. In the future, we will extend this work to inter-neighbor-group resource discovery and allocation for very large distributed dynamic real-time systems.

References

[1] Available at: http://www-csag.ucsd.edu/projects/agileO.html

[2] J. Basney, M. Livny, *Improving Goodput by Co-scheduling CPU and Network Capacity,* International Journal of High Performance Computing Applications, Vol. 13, No. 3, Fall 1999.

[3] J. Basney, M. Livny, P. Mazzanti, *Utilizing Widely Distributed Computational Resources Efficiently with Execution Domains*, Journal of Computer Physics Communications, Apr. 2000.

[4] F. Bonomi, A. Kumar, *Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler,* IEEE Transactions on Computers, Vol. 39, Iss. 10, pp.1232-1250, Oct. 1990.

[5] J. Cao, D. J. Kerbyson, G.R. Nudd, *Performance evaluation of an agent-based resource management infrastructure for grid computing,* 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, Brisbane, Qld., Australia, pp.311-318, May 2001.

[6] S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw, *Resource Management in Legion,* in the journal Future Generation Computer Systems, vol. 15, no. 5-6, pp. 583-594, October 1999.

[7] K. Connelly, A. Chien, *Breaking the Barriers: High Performance Security for High Performance Computing,* New Security Paradigms Workshop, Sept. 2002.

[8] K. Connelly, A. Chien, *Making High Performance Components Safe Enough,* Supercomputing, Nov. 2002.

[9] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy, A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation, Intl Workshop on Quality of Service, pp. 27-36, June 1999.

[10] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations,* International J. Supercomputer Applications, 15(3), 2001.

[11] I. Foster, A. Roy, V. Sander, *A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation*, 8th International Workshop on Quality of Service, (IWQoS 2000), pp. 181-188, June 2000.

[12] A. Grimshaw, Wm. A. Wulf, *Legion – A View From 50,000 Feet,* Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing, Los Alamitos, CA, Aug. 1996.

[13] K. Jun, L. Boloni, K. Palacz, D.C. Marinescu, *Agent-based resource discovery*, 9th Heterogeneous Computing Workshop, Cancun, Mexico, pp.43-52, May 2000.

[14] A. Iamnitchi, I. Foster, *On Fully Decentralized Resource Discovery in Grid Environments*, 2nd International Workshop on Grid Computing, Denver, Colorado, USA, pp.51-62, Nov. 2001.

[15] M. Livny, J. Basney, Raman, and T. Tannenbaum, *Mechanisms for High Throughput Computing*, SPEEDUP Journal, Vol 11, No. 1, June 1997.

[16] M. Maheswaran, *Data Dissemination Approaches for Performance Discovery in Grid Computing Systems,* In the Proceedings of 15th International Parallel and Distributed Processing Symposium, San Francisco, CA, USA, pp.23-27, Apr. 2001.

[17] A. Natrajan, M. Humphrey, A. Grimshaw, *Capacity and Capability Computing using Legion* International Conference on Computational Science, May. 2001.

[18] R. Raman, M. Livny, M. Solomon, *Matchmaking: Distributed Resource Management for High Throughput Computing,* In the Proceedings of 7th IEEE International Symposium on High Performance Distributed Computing, Chicago, IL, USA, pp.140-146, Jul. 1998.

[19] Rana, O.F. Bunford-Jones, D. Walker, D.W. Addis, M. Surridge, M. Hawick, K. *Resource discovery for dynamic clusters in computational grids*, Proceedings 15th International Parallel and Distributed Processing Symposium, San Francisco, CA, USA, pp.759-767, Apr. 2001.

[20] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, *RSVP: a new resource reservation protocol,* IEEE Networks Magazine, vol. 31, No. 9, pp. 8-18, September 1993.

Appendix C

Using Overlay Networks to Resist Denial-of-Service Attacks

Ju Wang and Andrew A. Chien
Department of Computer Science and Engineering
University of California, San Diego
{jwang,achien}@cs.ucsd.edu

Abstract

Proxy-network based overlays have been proposed to protect Internet applications against Denial-of-Service (DoS) attacks by hiding an application's location. We develop a formal framework which models attacks, defensive mechanisms, and proxy networks. We use the framework to analyze the general effectiveness of proxy network schemes to protect applications. Using our formal model, we analytically characterize how attacks, defensive schemes, and proxy network topology affect the secrecy of application location and general resource availability. Our results provide guidelines for the design of proxy networks; the formal framework provides a tool to study problems in this area.

Our analysis shows that proxy networks are a feasible approach to prevent infrastructure-level DoS attacks. Proxy network depth and system reconfiguration are the keys to achieving location hiding. Proxy network topology also has an important impact -- rich connectivity in the proxy network, a virtue in other circumstances, reduces effectiveness in location hiding. Finally, to avoid resource depletion, reactive resource recoveries are insufficient; proactive schemes are needed.

Keywords

security, availability, Denial-of-Service, overlay network

1 INTRODUCTION

Denial-of-service (DoS) attacks are a major security threat to Internet applications. Since 1998, there have been a series of large-scale distributed DoS attacks which effectively shut down popular sites such as Yahoo! and Amazon and the White House website was forced to move to a different location [1-5]. These attacks have serious economic impact and political repercussions, and may even threaten critical infrastructures and national security [6-8].

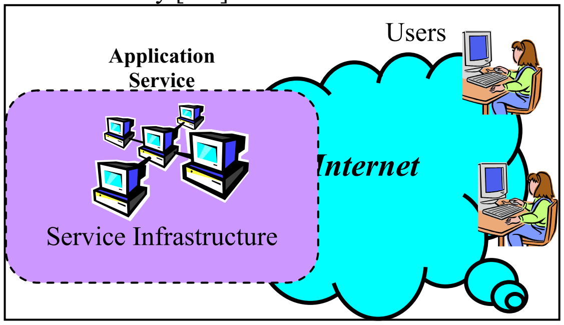


Figure 1 Example of Internet Application

In a Denial-of-Service attack, attackers can make the victim application unavailable to legitimate users by overloading the application with floods of network traffic or large amount of workload. DoS attacks can be categorized as *infrastructure level* or *application level attacks*. shows a typical Internet application deployment. The application service runs on a set of interconnected hosts, which is the service infrastructure; users access it via the Internet. *Infrastructure-level attacks* overload the service infrastructure, for example, by sending packet floods to saturate the victim network. In this case, attackers can effectively DoS an application without any knowledge of it except for its IP address. *Application-level attacks* cause denial-of-service by requesting large amounts of work at the application level or by exploiting weaknesses in the application.

Many Internet applications are publicly accessible, so they are easy targets for infrastructure-level DoS attacks. We are exploring the use of overlay proxy networks to tolerate infrastructure-level attacks. The key idea is to hide the Internet applications behind a proxy network, which is an overlay network (Figure 2). All accesses to the applications are mediated through the proxy network. Since only application level traffic can pass through the proxy network, infrastructure level attacks are no longer possible as long as the IP address of the application can be securely hidden. Furthermore, the proxy network needs to run on a large resource pool and be highly distributed and fault tolerant, so it can by itself tolerate DoS attacks and shield applications. The essence of this approach is the following. It is hard to make general applications highly distributed and DoS resistant. Therefore we build proxy networks with such capabilities, which are easier to build and can be shared among applications. We use them to shield the applications. Mechanisms such as Network Address Translation (NAT) can also hide the application's location. However, NAT boxes are vulnerable to DoS attacks, so they cannot shield applications; we will need networks of NATs to resist attacks. Current NAT technology does not scale up to support this. Our proxy networks provide a possible form of distributed NAT network.

A key capability of proxy networks is location-hiding, which is a component of a complete solution to DoS attacks. It provides a "safety period", during which an application's location is kept secret, and infrastructure-level attacks are prevented. It can be combined with other mechanisms such as application reconfiguration, redeployment, or even mobility to effectively protect applications against infrastructure-level attacks. If applications can change their location within a safety period, they can avoid DoS attacks indefinitely. However, there is a high cost to reconfigure applications; therefore there is a strong benefit to have effective location-hiding schemes that can provide long safety periods, reducing overhead and frequency of application reconfigurations. This paper studies the capability of proxy networks to provide effective location-hiding.

We are not the only researchers exploring the use of proxy networks for enhancing application security. Others, such as Secure Overlay Services (SOS) [9] and Internet Indirection Infrastructure (i3) [10], use existing overlay networks such as Chord [16] to hide the IP addresses of important nodes. To date, we know of no modeling or effectiveness analysis of these approaches. Our analysis considers a general class of proxy networks for location-hiding (SOS [9] and i3 [10] are instances of the class), and provides an understanding of what capabilities are feasible, and the importance of different elements of proxy network design. We believe such analysis of proxy network capabilities can lead to better understanding and design guidelines for the whole class of location-hiding approaches.

In this paper, we build a formal model to characterize proxy networks, attacks, and defensive mechanisms including proxy network reconfiguration and resource recovery schemes. We use this model to study the effectiveness of the proxy network approach. Using the model, we characterize the difficulty for attackers to penetrate the proxy network and discover application location. We also characterize how quickly resources can be compromised and the effectiveness of resource recovery policies such as intrusion detection-based reactive schemes and proactive schemes that do not rely on detection. Our study leads to the following qualitative conclusions:

1. Proxy networks with random proxy migration can effectively hide applications' IP addresses; thereby preventing infrastructure-level DoS attacks.
2. Proxy network depth and internal reconfiguration are critical to preventing attackers' penetration.
3. The topology of proxy networks is important. Surprisingly, rich connectivity, a virtue in other circumstances, can reduce a proxy network's ability to hide application location.
4. Reactive techniques for resource recovery are insufficient by themselves to avoid resource depletion. However, proactive schemes can successfully prevent resource depletion.

The model and qualitative results provide insights into how proxy networks should be designed to effectively protect applications from DoS attacks – either by hiding their location or protecting against resource depletion. Our study intends to build a better understanding of overlay networks' capability of location-hiding for DoS attack resistance, provide intuitions of how proxy networks should be designed, and build step stones for future studies based on more complex and realistic models in this area, rather than immediately and completely solve the DoS problem.

The remainder of the paper is structured as follows. Section 2 formulates the DoS problem and introduces our analytical models. Analytical results, insights and discussions are presented in Section 3. Section 4 discusses the implications of our analysis. Section 5 relates our work to the other studies, and then we conclude in Section 6 with a summary and a description of directions for future work.

2 ANALYTICAL MODEL

In this section, we develop an analytical model for the system. First, we give an overview of the proxy network scheme. Second, we describe the key components, including the resource pool, the proxy network, the attacks and the related defensive mechanisms. Third, we propose an analytical model to characterize these components. This model is used in Section 3 to study the DoS problem.

2.1 Proxy network scheme

Infrastructure-level DoS attacks target at the IP addresses of the victim applications. Today's Internet applications publish their IP addresses (for example via DNS) for convenient user access via the Internet, but their published IP addresses become obvious targets in DoS attacks. We use a proxy network approach to address this problem. In our approach, applications do not publish their IP addresses, instead, they hide behind a proxy network, an overlay network that runs on a resource pool of Internet hosts. The proxy network hides the IP addresses of all the nodes inside (including internal proxies and applications); only proxies at the edge publish their IP addresses (see Figure 2). All accesses to the applications are mediated by the proxy network via edge proxies. No one can easily discover the applications' location, thereby preventing infrastructure-level attacks.

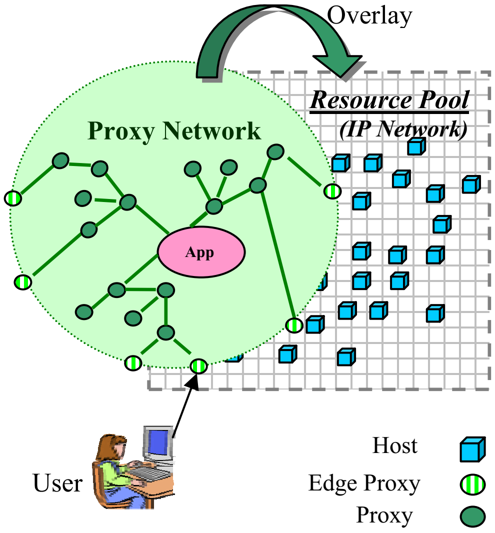


Figure 2 Proxy Network Scheme

There are two key challenges in the proxy network scheme. First, the proxy network should hide applications' IP addresses securely. Second, the proxy network itself should be resilient to DoS attacks, so it can shield the applications. The second challenge is more straightforward; proxies can be built as simple elements without persistent state. Without a need for strong consistency, replication schemes can be used to tolerate DoS attacks. In this paper, we focus on the first problem – location-hiding.

2.2 Resource Pool and Proxy Network

Before discussing the attacks and the defensive mechanisms, we formally describe the resource pool and the proxy network, and introduce a rigorous terminology. For simplicity, we study the case where there is only one application. We believe that our analysis can be extended to multiple applications sharing the same proxy network, but work is beyond the scope of this paper.

The resource pool consists of hosts in the Internet. We assume that the hosts can communicate directly if they have each other's IP address, and each host is identified by a unique IP address. A *node* in the overlay network is either a proxy or the application. When a node runs on a host, that host (or its IP address) is called the *location* of the node. We assume each node has a unique location at any moment (an injective mapping from nodes to hosts). Two nodes are *adjacent* if and only if they know each other's location. Obviously, adjacent nodes can communicate directly through the underlying hosts at the IP level. We use a *topology graph* to represent the overlay network. Vertices in the graph correspond to nodes in the overlay; edges correspond to the adjacency relationship. The minimum distance from edge proxies to the application in the topology graph is the *depth* of the proxy network. A conceptual view of a proxy network with depth 3 is shown in Figure 2. The topology graph describes the connectivity of the overlay network; two nodes can communicate at the overlay level if there is a path between them in the topology graph. More importantly, the topology graph also describes how the location information is shared among the overlay nodes, a critical aspect of how securely the proxy network can hide the application's location, because when attackers compromise a proxy node, they can locate all adjacent nodes.

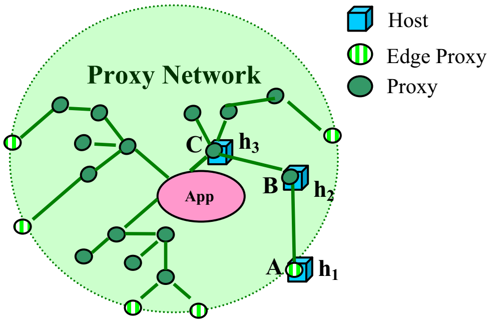


Figure 3 Proxy Network Penetration

2.3 Attacks

We focus on the use of proxy networks for location hiding. Therefore, the most important issue is *host compromise* attacks, which can penetrate the proxy network and reveal an application's location. Other attacks are considered in Section 4. In a successful host compromise attack, attackers can temporarily control the victim host and steal information from it. A host under such impact is considered *compromised*; otherwise it is *intact*. We overload the term *compromised* – a proxy is *compromised* if it runs on a compromised host.

At the overlay network level, host compromise attacks can reveal the location of overlay nodes. For example, in Figure 3, when proxy A is compromised, attackers *expose* the location of proxy B. Repeating this process, attackers penetrate the proxy network and may eventually cause *application exposure*, where the application is exposed to attackers.

At the resource pool level, host compromise attacks can cause resource loss. Unless those compromised hosts are recovered, they can no longer be used as intact resources. Host compromise attacks can eventually lead to *resource depletion*, where intact hosts in the resource pool are insufficient for proxy networks to operate correctly.

Attackers can either act autonomously (uncoordinated attacks) or cooperate (coordinated attacks).

2.4 Defensive mechanisms

We have two defensive mechanisms, each of which corresponds to the key risks, application exposure and resource depletion. At the overlay network level, proxy network reconfiguration mechanisms disrupt attacker penetration, thereby helping to prevent application exposure; at the resource pool level, resource recovery/reset mechanisms convert compromised hosts to intact state, helping to avoid resource depletion.

Proxy Network Reconfiguration

Proxy network reconfiguration mechanisms dynamically change proxies' location or structure of proxy networks, disrupting attacker penetration by invalidating location information exposed by attacks.

In this paper, we study random proxy migration, a simple form of proxy network reconfiguration. Proxies randomly change their location inside the resource pool, but do not change the topology of the proxy network. For example, in Figure 3, when proxy B migrates from h_2 to another host, it will notify its neighbors A and C of its new location. With random proxy migration, proxies can move to new locations unknown to attackers, therefore disrupting attackers' penetration. For example, suppose attackers exposed proxy B when B is on host h_2. When B migrates to another host, attackers' information about B becomes invalid (if both A and C are intact)[1]. Attackers cannot proceed unless they can discover B's current location. In addition, proxy migration can move proxies from compromised hosts to intact hosts. We study the effectiveness of such schemes to prevent application exposure.

Resource Recovery/Reset

Resource recovery/reset mechanisms at the resource pool level convert compromised hosts to the intact state. There are two triggering policies, reactive recoveries and proactive resets[2]. In reactive recoveries, compromised hosts are only recovered after compromise is suspected or detected. Proactive resets do not depend on detection, and reset hosts into the intact state regardless of their current state. Examples of proactive resets include timer-triggered reloading of hosts with clean and up-to-date system images, updating and creating new credentials,

[1] Migration of B is only effective when Proxy A and C are not compromised at the moment. Both resource recovery/reset and proxy reconfiguration can get proxies out of compromised state by either recovering the compromised host or moving the proxy to an intact host. It is considered in our analysis.

[2] "Reset" and "Recovery" here do not imply going back to a previous state. They set the hosts into a known clean state with all the known security holes fixed. Therefore, future attackers cannot easily compromise them through the known security holes.

and so on. We study the effectiveness of both schemes to prevent resource depletion.

2.5 Stochastic Models

Model of host compromise attack

We model occurrences of successful host compromises by one attacker as a Poisson process with rate ς; ς is the compromise speed and $\frac{1}{\varsigma}$ is the average time to compromise a host. To keep the model concise and simple, we assume hosts in the resource pool are widely distributed and do not have highly correlated vulnerabilities, so that one host compromise does not increase the speed of other compromises, even though the attackers are coordinated. Therefore we use the same compromise speed ς for all attacks. The probability of compromising a host within time t is given by $(1\,4\,e^{4\varsigma t})(t\,\varnothing 0)$.

Model of proxy network reconfiguration

Proxies randomly migrate in the resource pool. Occurrences of migration events on any specific proxy are modeled as a Poisson process with rate σ_r. All proxies migrate independently at the same rate. Mathematically, the probability of a proxy migrating within time interval t is $1\,4\,e^{4\sigma_r t}(t\,\varnothing 0)$.

Model of reactive recovery

Key attributes of reactive recoveries are true positive ratio and recovery delay. True positive ratio is the ratio of compromises that are eventually detected. Recovery delay is measured from the moment of compromise to the moment of recovery (if the compromise is eventually detected). In our model, the true positive ratio is ψ, and the expected recovery delay is $\frac{1}{\sigma_d}$.

Our model does not impose any specific distribution on the behavior of recovery. Any distribution capturing these two attributes (ψ and $\frac{1}{\sigma_d}$) will converge to our result. In this paper, for the mathematical convenience, a scaled exponential distribution is used. The probability of a reactive recovery within time t is given by $\psi(1\,4\,e^{4\sigma_d t})(t\,\varnothing 0)$.

Model of proactive reset

We model proactive reset events on a host as a Poisson process at rate σ_s. In other words, the average interval between two resets on a host is $\frac{1}{\sigma_s}$, and the probability of a proactive reset within time t is given by $(1\,4\,e^{4\sigma_s t})(t\,\varnothing 0)$.

Table 1 Notations of Analytical Model

Notation	Meaning
ς	Speed of host compromise
σ_r	Rate of proxy migration
$1/\sigma_d$	Expected delay of reactive recovery
ψ	True positive ratio of reactive recovery
σ_s	Rate of proactive reset

Our notations are summarized in Table 1.

Discussion

We use Poisson process to describe host compromises because it can concisely characterize the system with one parameter "speed of compromise (ς)". The Poisson model is suitable for stochastic processes which are statistically independent of the past. When the hosts in our system are carefully maintained with all the known security holes fixed, Poisson model is a reasonable approximation. Earlier studies [11, 12] also showed that Poisson model can correctly characterize the behavior of software system with a small number of bugs. This further justifies of our model.

Because little is understood analytically about the behavior of systems under host compromise attacks, we have chosen to use simple models that enable analysis and can concisely characterize the key attributes of the system as well as build intuition. At present, complex models quickly become intractable and their results can be hard to interpret. As an initial step, we ignore many details of the system (for example correlated vulnerabilities among hosts) to make the analysis tractable. Even though this may be very different to reality, we believe our analysis still provides a fundamental understanding of the problem, which is essential to future study based on more complex and realistic models. Our study intends to build a step stone for better understanding of this problem, rather than completely solve the DoS problem.

3 ANALYTICAL RESULTS

Using the models defined in Section 2, we study the effectiveness of the proxy network scheme. We focus on the two forms of successful attacks described in Section 2.3:

- ***Application Exposure***: How much time will it take attackers to penetrate the proxy network and expose the application? How do different parameters, such as speed of host compromise, speed of resource recovery, rate of proxy migration and topology of proxy networks, affect the effectiveness of the scheme?
- ***Resource Depletion***: Under what circumstances it is possible to keep the majority of the hosts intact, so that the proxy migration scheme makes sense? How effective are the resource recovery schemes against host compromise attacks?

3.1 Application Exposure

In this section, we prove that without proxy network reconfiguration, proxy networks cannot securely hide the location of the application. Then we prove that with random proxy migration, there exists a class of proxy networks that can provide effective location-hiding. Then we study how different parameters affect the effectiveness of our scheme (with a specific proxy network topology) and provide design guidelines. Finally, we discuss the impact of proxy network topology. In this section, we assume that there are sufficient intact hosts in the resource pool. The validity of this assumption is studied in Section 3.2.

We define a proxy network to be effective if the expected time for attackers to expose the application grows exponentially with the depth of the proxy network. In other words, for effective schemes, adding resources can significantly improve security.

3.1.1 Location hiding

Result I: *Without reconfiguration, proxy networks cannot effectively hide the application's location.*

Proof of Result I:
Consider a path from an edge proxy to the application as shown in Figure 4. Let T_ς be the expected time of host compromises. Without reconfiguration, the location of proxies does not change, and the topology of the proxy network does not change. Attackers only need to compromise all the proxies on a path to the application. It is trivial that the expected time to penetrate a proxy network with depth d is dT_ς; the expected time to application exposure grows linearly with d. Result I follows directly.ư

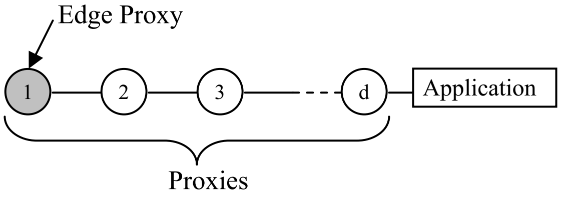


Figure 4 Path from edge proxy to application

Result II: *If the majority of the hosts in the resource pool are intact, with random proxy migration, there exists a class of proxy networks that can effectively hide the location of the application.*

Here is an intuitive explanation. Figure 4 shows a path from an edge proxy to the application; d is the length of that path. Initially only the edge proxy is exposed, and location of all the other proxies and the application is unknown to attackers. As described in Section 2.3, attackers can penetrate the proxy network starting from the edge proxy. If all the non-edge proxies (2 to d in the figure) can change their location periodically, then it can disrupt the penetration. For example, in Figure 4 if attackers managed to compromise proxy 2, then proxy 3 was exposed at that time. But this location information is only valid until proxy 3 migrates[3], and if attackers cannot compromise proxy 3 before that time, they cannot go any further. Intuitively, if the rate of proxy migration is higher than the speed of host compromise, it is hard for attackers to penetrate the proxy network, because proxies can almost always run away before they get compromised.

To prove Result II, we need Lemma3.1.1 and Proposition3.1.2.
Lemma 3.1.1: d is the depth of a proxy network with an arbitrary topology. ς is the speed of host compromise, $T_\varsigma=\varsigma^{-1}$ is the expected time of a host compromise. σ_r is the rate of proxy migration ($\sigma_r>2\varsigma$). When the majority of the hosts in the resource pool are intact, the expected time for any uncoordinated attacker to expose the application is between $\mathrm{N}((\frac{\sigma_r}{2\varsigma})^{d42})T_\varsigma$ *and* $\mathrm{N}((\frac{\sigma_r}{\varsigma})^{d41})T_\varsigma$.
Proof of Lemma 3.1.1 is in Appendix I.

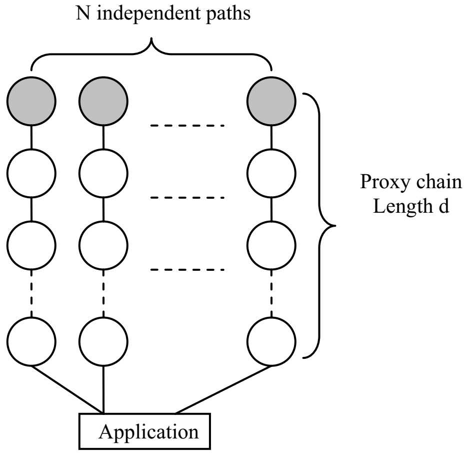


Figure 5 N independent proxy chains

Proposition 3.1.2: Consider a proxy network with a topology graph shown in , where there are N paths from edge proxies to the applications, and all the N paths are independent (vertex-disjoint). When the majority of the hosts in the resource pool are intact, the expected time for coordinated attackers to expose the application is between $\mathrm{T}(\frac{1}{N}(\frac{\sigma_r}{2\varsigma})^{d42})T_\varsigma$ *and* $\mathrm{T}(\frac{1}{N}(\frac{\sigma_r}{\varsigma})^{d41})T_\varsigma$ *(the meaning of σ_r, ς, d and T_ς is the same as in Lemma 3.1.1).*
Proof of Proposition 3.1.2 is in Appendix II.

Proof of Result II:
Lemma 3.1.1 shows that the expected time for uncoordinated attackers to expose the application grows exponentially with the depth of a proxy network. Therefore, with random proxy migration, proxy networks can effectively resist uncoordinated attacks to hide the application's location.
Proposition 3.1.2 shows that for a class of proxy networks shown in Figure 5, the expected time for coordinated attackers to expose the application grows exponentially with the depth of the proxy network. Therefore, there exist some proxy networks that can effectively resist coordinated attacks to hide the application's location. Result II follows directly. ư

To illustrate the effectiveness of the proxy network scheme, let us assume T_ς to be on the order of days[4]; namely, it may take attackers a few days to compromise a host. We consider a proxy network with depth 6. Without proxy network reconfiguration, we know from Result I that it will take about a few weeks to expose the application. With random proxy migration, if proxies migrate about once a few hours, then it will take attackers hundreds of years to expose the application.

[3] More precisely, if proxy 3 migrates after proxy 2 gets out of the compromised state (proxy 2 migrated to an intact host or proxy 2's host is recovered), attackers will lose track of proxy 3's location.

[4] We assume hosts in the resource pool are well maintained and they do not have known bugs. Attackers will need significant amount of time to discover and study new vulnerabilities, rather than using existing automated attack tools or worms. In reality, it may take hackers more than a few days (sometimes even weeks or months) to break into remote systems.

3.1.2 Parametric Analysis

In this section, we study how different parameters affect the effectiveness of the proxy network scheme. First Result III qualitatively describes the impact of different parameters. Then an extensive parametric study illustrates the impact of each parameter.

Result III: *Proxy migration rate and depth of proxy networks are the key factors to stop attackers' penetration; linear increase in the depth of the proxy network exponentially increases the time to application exposure.*

Proof of Result III:

Result III follows directly from Lemma 3.1.1 and Proposition 3.1.2. ư

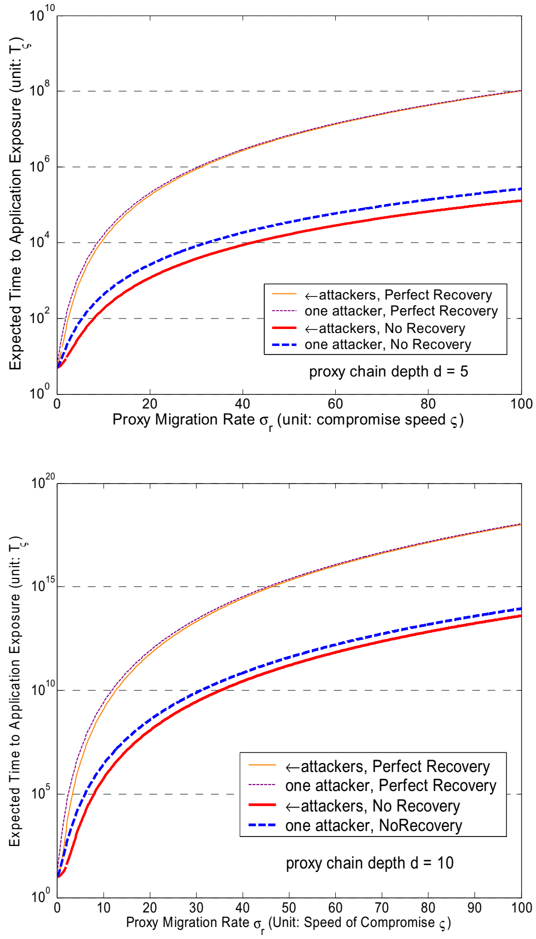


Figure 6 Impact of Proxy Migration Rate

To illustrate the impact of different parameters, such as proxy migration rate, proxy network depth, speed of resource recovery and number of coordinated attackers, we consider a proxy network with a linear chain topology, and plot the expected time to application exposure as a function of these parameters. To understand the impact of resource recovery schemes, we plot two boundary cases: no recovery and perfect recoveries, which immediately recover a host after its compromise. To understand the impact of coordinated attackers, we plot the two boundary cases "←attackers" and "one attacker". "← attackers" corresponds to the highest penetration speed attackers can achieve on a linear chain with sufficiently many coordinated attackers. "One attacker" corresponds to the case of a single attacker. Therefore the plots provide a set of envelopes for general cases (different resource recovery schemes and any coordinated attacks).

Figure 6 shows how the proxy migration rate affects the expected time to application exposure (d=5 and d=10 respectively as shown in the two graphs; d is the depth of the proxy network). From Figure 6 we can clearly see the trend that the expected time to application exposure significantly increases as migration rate increases (note that Y-axis is log scale). In fact, it increases at a polynomial rate with d as the exponent. For example, when d = 10, by doubling the migration rate, the time to exposure becomes three orders of magnitude longer.

Figure 7 shows the impact of proxy network depth. Similar to Figure 6, the plots are the boundary cases. The expected time to application exposure increases exponentially as the depth increases. For attackers, it means that each step further into the proxy network becomes exponentially harder than all the work they did before; and will quickly become intractable when depth gets fairly large. To illustrate the speed of growth, suppose attackers can compromise a host in a day, and proxies migrate 10 times a day. To penetrate a proxy network with a depth of 4 may take a few years; a depth of 6 may take a few hundred years; a depth of 10 may take a few million years, which practically means it will never happen. Therefore, depth of the proxy network is an effective barrier to stop attackers' penetration.

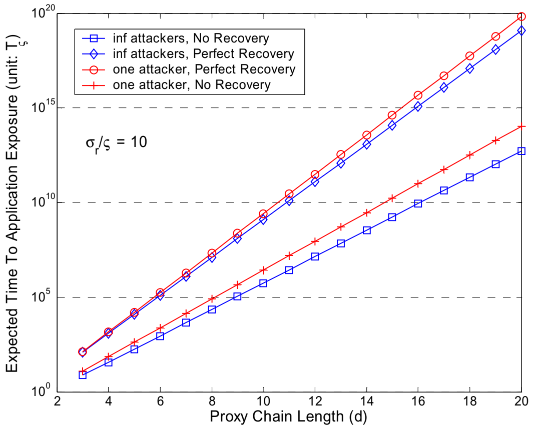


Figure 7 Impact of Proxy Chain Depth

From Figure 6 and Figure 7, we can see that both the proxy network depth and the proxy migration rate have significant impact on the effectiveness of the proxy network scheme. The depth of the proxy chain is the most dominant factor.

Resource recovery schemes and the number of coordinated attackers have limited impact on the overall security. By adjusting the proxy migration rate or the proxy chain depth, we can amortize the negative impact coming from those sources, as long as the majority of hosts are intact in the resource pool. However, this result does not imply good resource recovery schemes are unnecessary. Good recovery schemes are certainly favorable as shown in Figure 6 and

Figure 7. With better resource recoveries, we can use a smaller proxy network depth or a lower migration rate to achieve same level of security more efficiently. More importantly, as discussed in Section 3.2, the resource recovery schemes have unique impact at the resource pool level.

3.1.3 Impact of Proxy Network Topology

The topology of proxy networks is important. As discussed above, the depth of proxy networks is a dominant parameter. Besides that, the connectivity of proxy networks also has significant impacts on how much parallelism attackers can exploit to speed up application exposure. Previous discussion is based on a specific class of proxy network topology (Figure 5), where all paths from the edge proxies to the application are independent (Claim 3.1.3). For general topology, we have the following result.

Result IV: *Rich connectivity increases the penetration probability for attackers and shortens time to application exposure. If the connectivity is sufficiently high, the proxy network can no longer effectively hide the location of the application.*

Intuitively, in a richly connected topology, there are more paths leading to the application. Therefore, there is more parallelism attackers can exploit. Furthermore, vertex degree (number of edges that touch the vertex) is typically high in a richly connected topology. That favors attackers, because compromising one proxy can expose a large number of proxies.

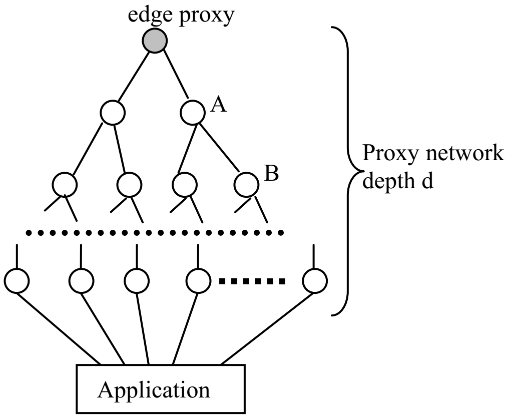


Figure 8 A proxy network topology (R=3)

A complete formal proof involves deep mathematic theory. It is beyond the scope of this paper, and will be addressed in our future work. Here we give an informal proof for a special case. Consider a regular graph (Figure 8), where all vertices (except the edge proxy and the proxies adjacent to the application) have degree R. Attackers' penetration can be considered as a series of retrials, with the edge proxy as the starting point. A trial succeeds if attackers reach depth d. The penetration probability depends on the probability of success in each trial. We study one such trial and show how vertex degree R affects this probability. For simplicity, we only consider the case with perfect resource recoveries.

Mathematically, this problem is a branching process [13]. Consider any pair of adjacent proxies, for example A and B in Figure 8. Let q be the conditional probability of B being eventually compromised if A is compromised. Without retrials, it is straightforward to prove that $q \mid \frac{\varsigma}{\varsigma 2 \sigma_r}$. Applying results in [13], we can compute the penetration probability.

Figure 9 plots how vertex degree affects the penetration probability. It shows that proxy networks with higher vertex degrees are easier to be penetrated.

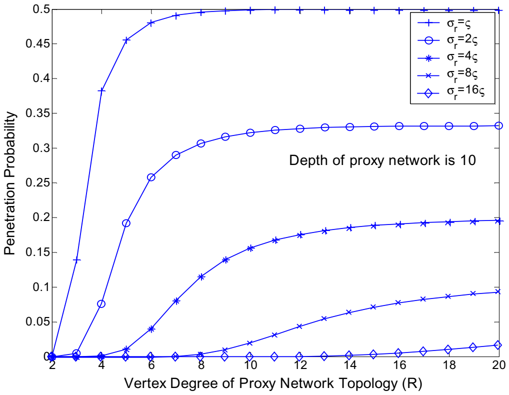


Figure 9 Impact of Vertex Degree

Furthermore, from the properties of branching processes [13], we know that q(R-1) is a criticality metric. It is qualitatively different on each side of the critical points. In the sub-critical case (q(R-1)<1), the depth of the proxy network is a theoretical barrier to stop penetration; the probability to penetrate a large depth can be arbitrarily small. On the other hand, in the super-critical case (q(R-1)>1), the depth of the proxy network is no longer an effective barrier to stop attackers' penetration; attackers can reach any depth with a non-trivial probability if given enough time. The topologies discussed in previous sections are in fact in the sub-critical case. When choosing a proxy network topology, the sub-critical case is more favorable.

To summarize Section 3.1, we first proved it necessary to have some form of reconfiguration mechanisms in the proxy network scheme. Then we proved that our proxy network scheme with random proxy migration can effectively prevent attackers' penetration and securely hide the application's location. With appropriate proxy network topologies, the depth of the proxy network is a dominant factor to the overall security, and the proxy migration rate have a significant impact. Choosing these parameters appropriately can effectively stop attackers' penetration. Topology of the proxy network is also important. Rich connectivity can increase the penetration probability and can qualitatively reduce the effectiveness of the proxy network scheme.

3.2 Resource Depletion

All the previous discussions are based on the assumption that we can somehow keep the majority of the hosts intact. This section studies the validity of this assumption.

Result V: *Reactive recoveries alone are insufficient to avoid resource depletion.*

Result VI: *When proactive resets, which do not rely on detection, are used, it is possible to keep the majority of hosts intact in the resource pool.*

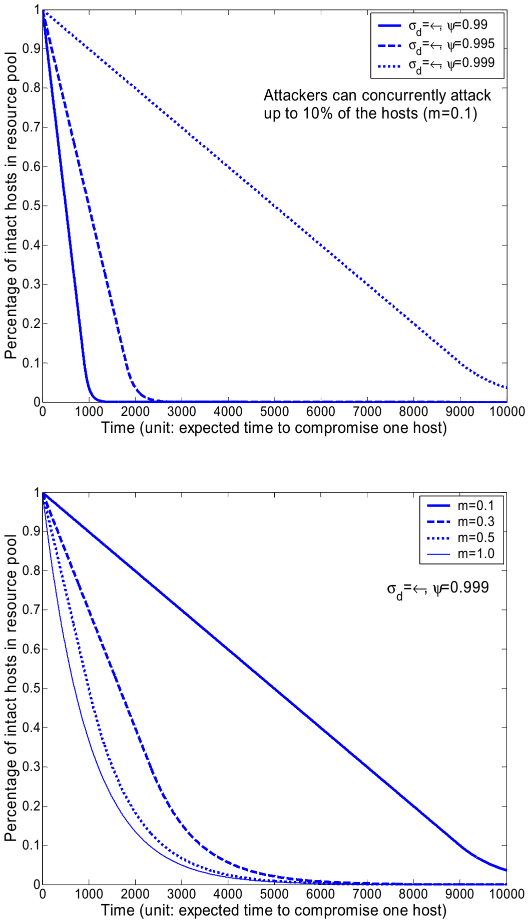


Figure 10 Resource depletion w/o proactive reset

Lemma 3.2.1: Assume initially all hosts are intact. Let m be the percentage of hosts attackers can concurrently attack, and f(t) be the expected percentage of intact hosts in the resource pool. We know

$$\begin{cases} f(t) \varnothing C_2 & when \quad m\, \Omega C_1 \\ \lim_{t \Downarrow \leftarrow} f(t) \mid C_1 & when \quad m \} C_1 \end{cases}$$

$$where \begin{cases} C_1 \mid (1 2 \frac{\varsigma \psi}{\sigma_d\ 2\ \sigma_s} 2 \frac{\varsigma(1 4\ \psi)}{\sigma_s})^{41} \\ C_2 \mid 1 4 \frac{\varsigma \psi}{\sigma_d\ 2\ \sigma_s} m\, 4 \frac{\varsigma(1 4\ \psi)}{\sigma_s} m \end{cases}.$$

Proof of Lemma 3.2.1 is in Appendix III.

Proof of Result V:

From Lemma 3.2.1, we can see that when there are only reactive recoveries (σ_s=0), $\lim_{t \Downarrow \leftarrow} f(t) \mid C_1 \mid C_2 \mid 0$ if ψ<1. In practice, ψ is always less than 1; therefore all hosts will eventually be compromised in this case. Result V is proved.ư

Intuitively, because not all intrusions are detected (ψ<1), reactive recoveries cannot recover all the compromised hosts; the residues accumulate over time and eventually cause resource depletion. Figure 10 shows that even if we have almost perfect detectors, which can detect almost all compromises (>99%) and instantaneously recover all the detected compromises, the percentage of intact hosts still drops fairly fast and eventually goes zero. It is worse when attackers can attack more hosts concurrently or the resource pool is smaller.

Proof of Result VI:

When the resource pool is sufficiently large such that mΩC$_1$, from Lemma 3.2.1, we know that the percentage of intact hosts is always higher than C$_2$. By appropriately choosing σ_s, σ_d, ψ and the size of the resource pool, C$_2$ can be arbitrarily close to 1. Namely, the majority of the hosts are intact in the resource pool. Therefore Result VI is proved.ư

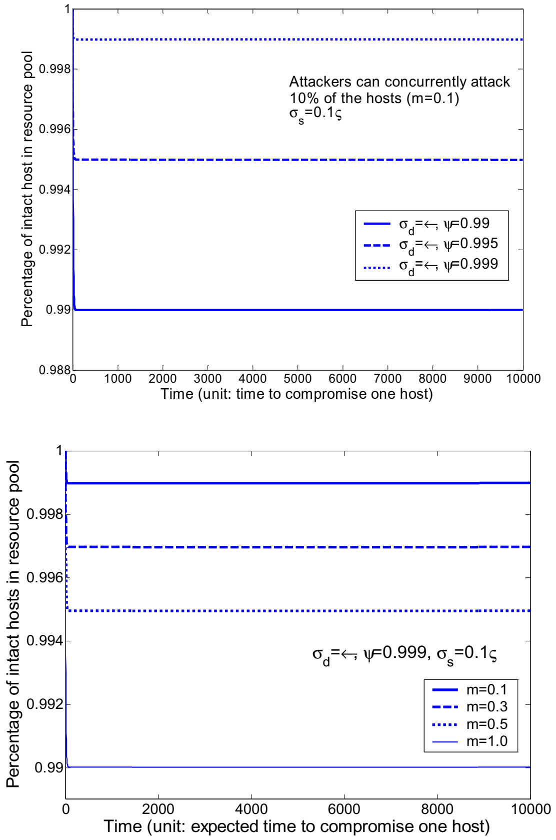


Figure 11 Resource availability with proactive reset

Figure 11 shows the impact of proactive reset. A proactive reset at a low rate (10 times slower than speed of compromise) is added to the cases in Figure 10. This small input fundamentally changed the system behavior. Now the percentage of intact hosts stabilizes at a number close to 1. Namely, the resource pool can keep almost all of the hosts intact over infinite time. This proves the need for proactive schemes that do not rely on detection.

shows that it is possible to avoid resource depletion with proactive resets. The Y-axis is the percentage of intact hosts in a stabilized system. It plots the worst case where attackers can concurrently attack all the hosts in the resource pool (m=1). It shows that even if no reactive recoveries are used

(σ_d=0, ψ=0), proactive resets can still keep a significant percentage of hosts intact. With reactive recoveries, proactive resets at fairly low rates can keep most hosts intact. The reactive recoveries shown in are realistic. Results from [14, 15] show that state of art intrusion-detection systems can achieve better than this.

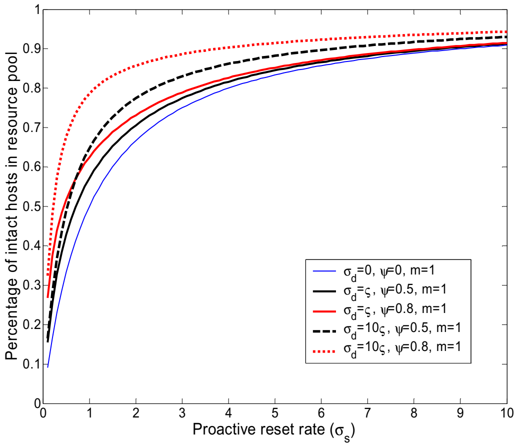


Figure 12 Proactive reset rate vs. intact host percentage

3.3 Summary

In this section, we studied the effectiveness of the proxy network scheme, and focused on two threats in particular: application exposure and resource depletion. We proved it necessary to have reconfiguration in the proxy network to hide application's location. Then we proved that with random proxy migration, our proxy network scheme can securely and effectively hide the application's location; we also proved that it is possible to keep most hosts in the resource pool intact. Combining these results, we have shown that our proxy network scheme is a feasible approach to securely hide application's location, thereby preventing infrastructure-level DoS attacks.

In our study, we also derived following design guidelines.

1. Proxy network depth is a dominant factor to location-hiding; proxy migration rate also has significant impact. With appropriate proxy network topology, these two factors can effectively stop attackers' penetration.
2. The topology of proxy networks is important. Surprisingly, rich connectivity, a virtue in other circumstances, may reduce the effectiveness of proxy networks.
3. Reactive resource recoveries are insufficient by themselves to avoid resource depletion. Proactive schemes that do not rely on detection are necessary.

4 DISCUSSION

We have proved that the proxy network scheme is a feasible approach to location-hiding. Our results have several implications to similar approaches that use overlay networks to hide the location of important nodes (secret nodes).

First, such overlay networks need to have some form of reconfiguration to prevent attackers' penetration. Without it, the approach is fundamentally vulnerable to host compromise attacks. Current approaches, such as SOS [9] and i3 [10], which do not have any active reconfiguration mechanisms in the overlay network, have this weakness.

Second, the secret nodes should be placed at the core of the overlay network, far away from the edge nodes in the topology graph. Caching the IP address of overlay nodes to shorten the route between overlay nodes, as suggested in i3 [10], can decrease the depth of the overlay network, therefore severely undermine the effectiveness of the scheme.

Third, the overlay network should have the least connectivity necessary to maintain good connection between the edge nodes and the secret nodes. General purpose overlays, such as Chord [16], that have high vertex degrees, may not be suitable.

We have shown that rich connectivity is not favorable for security. But good connectivity is necessary to tolerate failures, keeping applications reachable from users. So there is a balance between security and failure tolerance. How to choose an optimal topology is part of our future work. This paper only brings up the point that more connectivity in the proxy network is not always good, and warns against careless use of existing overlays, such as Chord, that are designed for completely different purposes.

Last, to maintain a resource pool of hosts, intrusion detection-based reactive recoveries alone are insufficient. Routine maintenance and occasional resets are critical over a long period of time. Furthermore, all the hosts in the resource pool need to be carefully and independently administrated, and regularly updated with security patches, so that they are less likely to share vulnerabilities. Otherwise, when hosts have correlated vulnerabilities, attackers can easily compromise a large number of hosts in a short while; it can greatly undermine the effectiveness of the scheme. For this reason, collecting home PCs scattered in the Internet to construct a resource pool may not be appropriate.

We only studied host compromise attacks in this paper. Other attacks can also discover application's location. But they are not major threats to the system. We briefly describe them below.

- Traffic analysis on the proxy network at Internet scale helps attackers to locate the application. But we do not consider it as a realistic threat, because it requires a prohibitive amount of resource and cooperation from major ISPs.
- Espionage on secret configuration and deployment information of the proxy network also helps attackers to locate the application. We resort to appropriate administrative policies and legislative means to prevent this type of attacks and punish the perpetrators.

5 RELATED WORK

Effectively resisting denial-of-service attacks is an important open problem. There are many ongoing studies, which can be categorized into two approaches: preventive and tolerant approaches. Preventive approaches try to stop or deter attacks from the source, which include Intrusion detection systems

[17-22], network ingress filtering [23] and IP trace-back schemes [24-27]. Tolerant approaches focus on mitigating the attack impact on the victim by means of system reconfiguration [28], resource isolation [28, 29] or load balance [30-33].

Many researchers are exploring the use of overlay networks to tolerate or avoid DoS attacks. The Secure Overlay Services (SOS) project [9] in Columbia University is one of them. They use Chord [16] in the overlay network to provide some amount of anonymity to hide the location of secret "servlets". There are primitive analytical results about the system security under simple attack models such as DoS attack on individual hosts. However, the analysis is tied to their Chord-based SOS design and they did not consider host compromise attacks, which are the main threat to their scheme. Internet Indirection Infrastructure (i3) [10] also suggested the use of Chord overlay network to hide the location of the application. They did not consider host compromise attacks and they did not fully analyze the effectiveness of their scheme. To our knowledge, our work is the first attempt of a thorough analysis in this area.

Here we have studied how to hide application location. Interestingly a complementary problem, hiding user identity, has been well studied since the early eighties. The solutions range from the early mix email server [34], to distributed Onion Routing schemes [35], and to the more recent Peer-to-Peer schemes such as Tarzan [36] and Pasta [37]. A key difference between the two problems is that there are many users in the system while there are only a handful of applications. Most of the schemes are based on the idea of mixing all input from all users so that an outsider cannot associate a particular message to a particular user. Another key difference is that user initiates the communication. In some schemes, such as Onion Routing [35], senders need to construct a route to the receiver before hand. These key differences make the two problems incomparable, and solutions in that area do not apply directly.

6 SUMMARY AND FUTURE WORK

We built a formal framework to rigorously study the properties of the system. Based on our analytical models, we have the following results.

1. Proxy networks with random proxy migration can effectively hide applications' IP addresses; thereby preventing infrastructure-level DoS attacks.
2. Proxy network depth and internal reconfiguration are critical to preventing attackers' penetration.
3. The topology of proxy networks is important. Surprisingly, rich connectivity, a virtue in other circumstances, can reduce a proxy network's ability to hide application location.
4. Reactive techniques for resource recovery are insufficient by themselves to avoid resource depletion. However, proactive schemes can successfully prevent resource depletion.

Future work includes the following.

1. More extensive study on the relationship between proxy network topology and security (hiding application's location) and failure resilience (maintaining connectivity) with the objective of guiding the design of an optimal proxy network topology.
2. Study of other forms of proxy network reconfigurations which achieve comparable levels of security at lower performance overheads.
3. Study of how (DoS or host compromise) attacks on one host affect other hosts in the resource pool, when hosts do share vulnerabilities (a new compromise model).

REFERENCE

1. CERT, *"Code Red II:" Another Worm Exploiting Buffer Overflow In IIS Indexing Service DLL*. 2001. http://www.cert.org/incident_notes/IN-2001-09.html

2. CERT, *"Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL*. 2001. http://www.cert.org/incident_notes/IN-2001-08.html

3. CERT, *CERT® Advisory CA-2003-04 MS-SQL Server Worm*. 2003. http://www.cert.org/advisories/CA-2003-04.html

4. Williams, M., *EBay, Amazon, Buy.com hit by attacks*. 2000. http://www.nwfusion.com/news/2000/0209attack.html

5. Fonseca, B., *Yahoo outage raises Web concerns*. 2000.http://www.nwfusion.com/news/2000/0209yahoo2.html

6. Miller, J., *2004 IT budget request focuses on homeland defense, cybersecurity*, in *Government Computer News*. 2003. http://www.gcn.com/vol1_no1/homeland/20903-1.html

7. Frank, D., *Cybersecurity called key to homeland defense*. 2001,FCW.COM. http://www.fcw.com/fcw/articles/2001/1001/news-cyber-10-01-01.asp

8. Schneider, F.B., *Trust in Cyberspace*. 1999: National Academy Press. 331.

9. Keromytis, A.D., V. Misra, and D. Rubenstein. *SOS: Secure Overlay Services*. in *ACM SIGCOMM'02*. 2002. Pittsburgh, PA: ACM.

10. Stoica, I., et al. *Internet Indirection Infrastructure*. in *SIGCOMM*. 2002. Pittsburge, Pennsylvania USA.

11. Littlewood, B., *Predicting software reliability.* Phil. Trans. R. Soc., 1989. **327**: p. 513-527.

12. Adams, E.N., *Optimizing preventive service of software products.* IBM Journal of Research and Development, 1984. **28**(1): p. 2-14.

13. Harris, T.E., *The Theory of Branching Processes*. 1963: Prentice-Hall Inc.

14. Lippmann, R.P., et al. *Evaluating Intrusion Detection Systems: the 1998 DARPA Off-Line Intrusion Detection Evaluation*. in *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*. 2000.

15. Lippmann, R., et al., *The 1999 DARPA Off-Line Intrusion Detection Evaluation*. 2000, MIT Lincoln Lab

16. Stoica, I., et al. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*. in *ACM SIGCOMM'01*. 2001.

17. Vigna, G. and R.A. Kemmerer, *NetSTAT: a network-based intrusion detection system.* Journal of Computer Security, 1999. **7**(1): p. 37-71.

18. Axelsson, S., *Intrusion Detection Systems: A Survey and Taxonomy*. 2000, Chalmers University of Technology: Goteborg, Sweden

19. Cowan, C., et al. *Automatic Detection and Prevention of Buffer-Overflow Attacks*. in *the 7th USENIX Security Symposium*. 1998. San Antonio, TX.

20. Kim, G.H. and E.H. Spafford, *Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection*. 1995, Purdue University

21. Kumar, S. and E.H. Spafford. *A Pattern Matching Model For Misuse Intrusion Detection*. in *Proceedings of the 17th National Computer Security Conference*. 1994.

22. Wagner, D. and D. Dean. *Intrusion detection via static analysis*. in *2001 IEEE Symposium on Security and Privacy*. 2001. Oakland, CA, United States: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy 2001..

23. Ferguson, P. and D. Senie, *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing.* The Internet Society, 1998.

24. Snoeren, A.C., et al. *Hash-based IP traceback*. in *ACM SIGCOMM 2001-Applications, Technologies, Architectures, and Protocols for Computers Communications-*. 2001. San Diego, CA, United States: Computer Communication Review. v 31 n 4 2001.

25. Song, D.X. and A. Perrig. *Advanced and authenticated marking schemes for IP traceback*. in *20th Annual Joint Conference of the IEEE Computer and Communications Societies*. 2001. Anchorage, AK, United States: Proceedings - IEEE INFOCOM. v 2 2001.

26. Stone, R. *An IP Overlay Network for Tracking DoS Floods*. in *the 2000 USENIX Security Symposium*. 2000. Denver, CO.

27. Savage, S., et al., *Practical network support for IP traceback.* Computer Communication Review, 2000. **30**(4): p. 295-306.

28. *Mutable Services*, New York University. http://www.cs.nyu.edu/pdsg/projects/mutable-services/mutable-services.htm

29. Spascheck, O. and L.L. Peterson. *Defending Against Denial of Service Attacks in Scout*. in *The 3rd symposium on operating systems design and implementation.* 1999.

30. Welsh, M., D. Culler, and E. Brewer. *SEDA: An Architecture for Well-Conditioned, Scalable Internet Services*. in *The 18th symposium on Operating Systems Principles*. 2001.

31. *Robust Networks*, Princeton University. http://www.cs.princeton.edu/nsg/robust/

32. *Websphere Edge Services Architecture*, IBM. http://www-3.ibm.com/software/webservers/edgeserver/doc/esarchitecture.pdf

33. *Network Load Balancing Technical Overview -- Microsoft Application Center*, Microsoft Corporation. http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/acs/Default.asp

34. Chaum, D.L., *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms.* Communications of the ACM, 1981. **24**(2): p. 84-90.

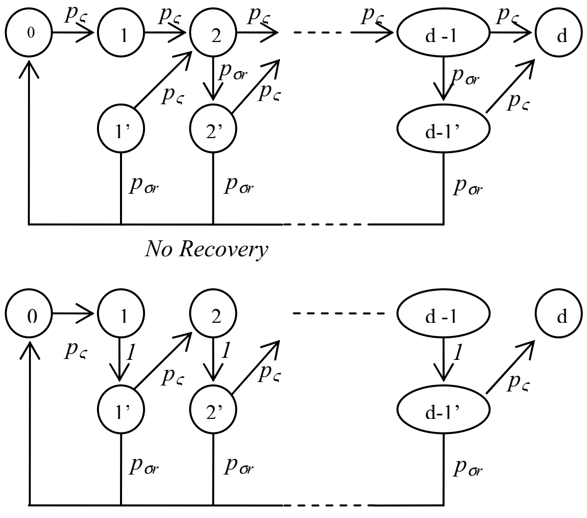
35. Reed, M.G., P.F. Syverson, and D.M. Goldschlag, *Anonymous Connections and Onion Routing.* IEEE Journal on Selected Areas in Communication Special Issue on Copyright and Privacy Protection, 1998.

36. Freedman, M.J., et al. *Introducing Tarzan, a Peer-to-Peer Anonymizing Network Layer*. in *1st International Workshop in Peer-to-Peer Systems (IPTPS'02)*. 2002. Cambridge, Massachusetts.

37. Elnikety, S., et al., *Pasta: Anonymous Peer-to-Peer Email System*. 2002, Rice University

APPENDIX I. PROOF OF LEMMA 3.1.1

In this case, we consider a single attacker who attacks one host at a time. Since no proxies run on same hosts at the same time, at most one proxy can be under attack at any moment in this scenario. We consider a path from an edge proxy to the application as shown in Figure 4. The Markov state transition graph is shown in Figure 13.



No Recovery

Perfect Recovery

Figure 13 Markov State Transition (One Attacker)

$p_ς$ is the probability of compromising one host within unit time ÷*t* ($p_ς^{-1}$ is the expected time of compromising a host); $p_{σr}$ is the probability of a proxy migration within unit time. In state 0, only the edge proxy is exposed. In state k (1ΩkΩd), the kth proxy is compromised. In state k' (1Ωk<d), the kth proxy is not compromised, but the (k+1)th proxy is exposed. We study the expected time from state 0 to reach state n in two boundary scenarios: no recovery and perfect recovery. When there is no recovery, a proxy will stay compromised until it migrates. With perfect recovery, hosts are instantaneously recovered (in the state transition graph, state k goes to state k' with probability 1).

(A) No Recovery

T$_k$ denotes the expected time to reach state n from state k (kΩd). Obviously, T$_d$ = 0 and we want T$_0$. It is straightforward to get a set of linear equations:

$$\begin{cases} T_0 \mid 1 2\ p_ς T_1\ 2\ (1 4\ p_ς)T_0 \\ T_1 \mid 1 2\ p_ς T_2\ 2\ (1 4\ p_ς)T_1 \\ T_{1'} \mid 1 2\ p_ς T_2\ 2\ p_{σr} T_0\ 2\ (1 4\ p_ς\ 4\ p_{σr})T_{1'} \\ T_k \mid 1 2\ p_ς T_{k21}\ 2\ p_{σr} T_{k'}\ 2\ (1 4\ p_ς\ 4\ p_{σr})T_k \\ T_{k'} \mid 1 2\ p_ς T_{k21}\ 2\ p_{σr} T_0\ 2\ (1 4\ p_ς\ 4\ p_{σr})T_{k'} \end{cases} \quad \text{(I)} \quad (1<k<d).$$

From (I), we get

$$\begin{cases} T_k \mid \dfrac{1}{p_ς} b_k\ 2\ b_k T_{k21}\ 2\ (1 4\ b_k)T_0 & (k \{ d) \\ b_k \mid \begin{cases} 1 & (k \mid 0) \\ \dfrac{1 2\ 2x}{(1 2\ x)^2} & (0 \{ k \{ n) \end{cases} \\ x \mid \dfrac{p_{σr}}{p_ς} \end{cases} \qquad \text{(II)}.$$

Solve (II), we have $T_0 \mid \dfrac{1}{p_ς}\dfrac{2y^{d41}\ 4\ y^{d42}\ 41}{(y 4 1)}$, where $y \mid \dfrac{(1 2\ x)^2}{1 2\ 2x}$.

(B) Perfect Recovery

Similar analysis can lead us to a set of linear equations in the same form as (II), but in this case $b_k \mid \dfrac{1}{1 2\ x}$. Solving it, we get

$$T_0 \mid \frac{1}{p_ς}\frac{(1 2\ x)^d\ 4 1}{x}.$$

Results from (A) and (B) holds for any unit time $\div t$. Therefore, we have $x \,|\, \lim_{\div t \Downarrow 0} \frac{p_{\sigma_r}}{p_\varsigma} \,|\, \frac{\sigma_r}{\varsigma}$ and $\lim_{\div t \Downarrow 0} \frac{1}{p_\varsigma} \,|\, \frac{1}{\varsigma} \,|\, T_\varsigma$. So $T_0 \,|\, \mathrm{N}((\frac{\sigma_r}{\varsigma})^{d41})T_\varsigma$ for perfect recovery and $T_0 \,|\, \mathrm{N}((\frac{\sigma_r}{2\varsigma})^{d42})T_\varsigma$ for no recovery. Lemma 3.1.1 is provedư

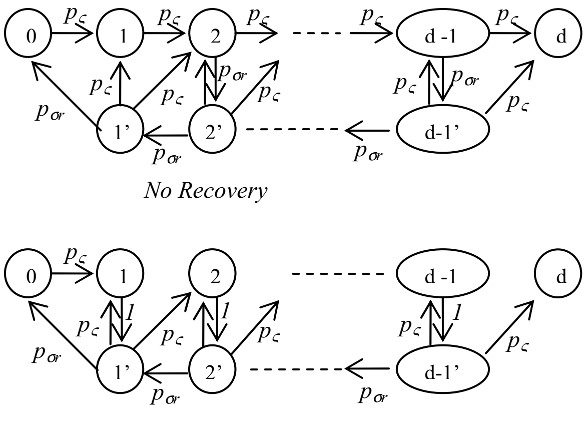
APPENDIX II. PROOF OF PROPOSITION 3.1.2

First we prove the following Lemma.

Lemma II.1: *Consider a proxy network with a linear chain topology as shown in* Figure 4. *Let d be the length of the chain. ς is the speed of host compromise, and $T_\varsigma=\varsigma^{-1}$ is the expected time of a host compromise. σ_r is the rate of proxy migration ($\sigma_r>2\varsigma$). When the majority of the hosts in the resource pool are intact, the expected time for coordinated attackers to expose the application is between* $\mathrm{N}((\frac{\sigma_r}{2\varsigma})^{d42})T_\varsigma$ *and* $\mathrm{N}((\frac{\sigma_r}{\varsigma})^{d41})T_\varsigma$ *.*

Proof of Lemma II.1:

In this case, we assume attackers can concurrently attack all the exposed proxies and the proxy network is a linear chain of proxies as shown in Figure 4. Markov state transition graph is shown in Figure 14. We use the same set of notation as in Appendix I.



No Recovery

Perfect Recovery

Figure 14 Markov State Transition (Multiple Attackers)

(A) No Recovery

From the state transition graph, we can get

$$\begin{cases} T_0 \,|\, 12\, p_\varsigma T_1 \,2\, (14\, p_\varsigma)T_0 \\ T_1 \,|\, 12\, p_\varsigma T_2 \,2\, (14\, p_\varsigma)T_1 \\ T_{1'} \,|\, 12\, p_\varsigma(T_2 \,2\, T_1) \,2\, p_{\sigma r}T_0 \,2\, (14\,2p_\varsigma \,4\, p_{\sigma r})T_{1'} \\ T_k \,|\, 12\, p_\varsigma T_{k21} \,2\, p_{\sigma r}T_{k'} \,2\, (14\, p_\varsigma \,4\, p_{\sigma r})T_k \qquad (k \} 1) \\ T_{k'} \,|\, 12\, p_\varsigma(T_{k21} \,2\, T_k) \,2\, p_{\sigma r}T_{k41'} \,2\, (14\,2p_\varsigma \,4\, p_{\sigma r})T_{k'} \end{cases}$$

Solve it, we get

$$T_0 \,|\, \frac{1}{p_\varsigma}(1\,2\, \frac{(\frac{x}{2})^{d41} \,4\,1}{(\frac{x}{2})\,4\,1} \,2\, \frac{(\frac{x}{2})^{d41} \,4\,1}{((\frac{x}{2})\,4\,1)^2}\frac{(\frac{x}{2})\,2\,1}{x\,2\,1}$$

$$4\,\frac{(\frac{x}{2})\,2\,1}{x\,2\,1}\frac{d\,4\,1}{(\frac{x}{2})\,4\,1})$$

where $x \,|\, \frac{p_{\sigma r}}{p_\varsigma}$.

(B) Perfect Recovery

With similar analysis, we get $T_0 \,|\, \frac{1}{p_\varsigma} \,2\, (1\,2\, \frac{1}{p_\varsigma})(\frac{x^d \,4\, x}{x\,4\,1}) \,2\, (2\,2\, \frac{1}{p_\varsigma})(\frac{x^d \,4\, x}{(x\,4\,1)^2} \,4\, \frac{d\,4\,1}{x\,4\,1})$, where $x \,|\, \frac{p_{\sigma r}}{p_\varsigma}$. With the same method used in Appendix I, we can get $T_0 \,|\, \mathrm{N}((\frac{\sigma_r}{\varsigma})^{d41})T_\varsigma$ for perfect recovery and $T_0 \,|\, \mathrm{N}((\frac{\sigma_r}{2\varsigma})^{d42})T_\varsigma$ for no recovery. ư

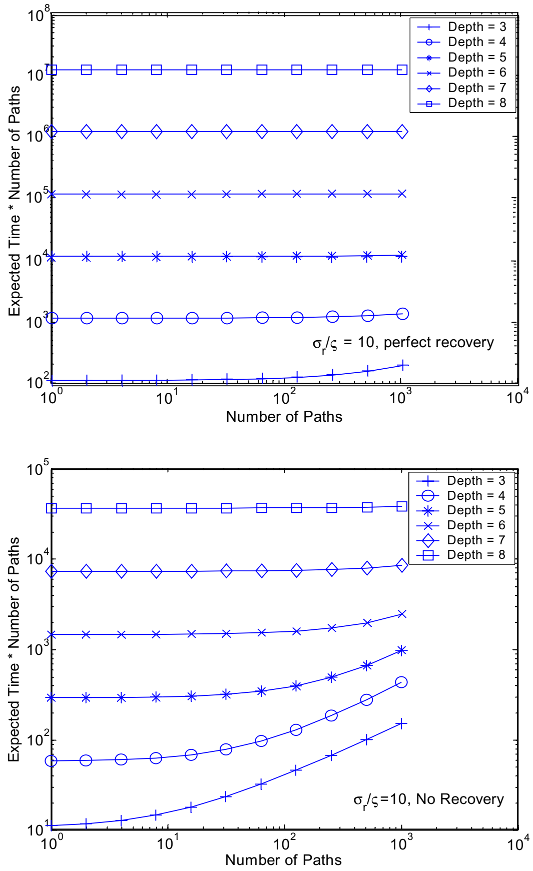


Figure 15 Impact of Independent Paths

Now we study a proxy network with topology shown in . A general proof is beyond this paper. Here we present a set of numerical results in Figure 15 to validate Proposition 3.1.2. From the Markov state transition graphs in Figure 14, we can obtain the transition matrix. Using this matrix, we numerically compute the expected time of application exposure. T_0 and T are respectively the expected time to application exposure when attackers penetrate from one path and when attackers penetrate from N independent paths. In Figure 15, the X-axis is number of paths N, and the Y-axis is N*T. From the figure we know that N*T∅T_0 for both boundary cases. We claim (without proof) that N*T∅T_0 is true for any general case within the boundary. Using Lemma II.1, we know that T_0 is between $\mathrm{N}((\frac{\sigma_r}{2\varsigma})^{d42})T_\varsigma$ and $\mathrm{N}((\frac{\sigma_r}{\varsigma})^{d41})T_\varsigma$. Therefore, Proposition 3.1.2 follows. ư

APPENDIX III. PROOF OF LEMMA 3.2.1

Figure 16 shows the state transition graph of hosts in the resource pool. f(t) denotes the expected percentage of intact hosts in the resource pool; g(t) denotes the expected percentage of the compromised hosts that can eventually be detected; and h(t) denotes the expected percentage of compromised hosts that can never be detected. m is the percentage of hosts concurrently attacked.

There are two cases: m>f(t), denoted as world A_1 and f(t) in this case is denoted by f_{A1}(t); mΩf(t), denoted as world A_2 and f(t) in this case is denoted by f_{A2}(t). In world A_1, attackers can concurrently attack all intact hosts; in world A_2, m bounds attackers' capability.

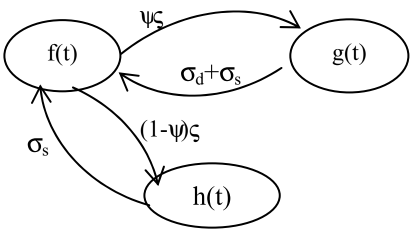


Figure 16 State Transition Graph

We first prove lemma III.1 and lemma III.2.

Lemma III.1:

$\lim_{t \Downarrow \leftarrow} f_{A_1}(t) \mid C_1$ *when* *f(0)+g(0)+h(0)=1,*

where $C_1 \mid (1\,2\,\frac{\varsigma\psi}{\sigma_d\;2\;\sigma_s}\,2\,\frac{\varsigma(1\,4\;\psi)}{\sigma_s})^{41}$.

Proof: From Figure 16, we can get the following differential equations:

$$\begin{cases} \frac{df(t)}{dt} \mid 4\varsigma f(t)\,2\,(\sigma_d\;2\;\sigma_s)g(t)\,2\;\sigma_s h(t) \\ \frac{dg(t)}{dt} \mid \psi\varsigma f(t)\,4\,(\sigma_d\;2\;\sigma_s)g(t) \\ h(t) \mid 1\,4\,f(t)\,4\,g(t) \end{cases}$$

Solve them, and we can get the following result:

$f_{A_1}(t) \mid A_1 e^{\pi_1 t}\;2\;B_1 e^{\pi_2 t}\;2\;C_1$, where $\pi_1<0$ and $\pi_2<0$ and $C_1 \mid (1\,2\,\frac{\varsigma\psi}{\sigma_d\;2\;\sigma_s}\,2\,\frac{\varsigma(1\,4\;\psi)}{\sigma_s})^{41}$. This result holds when f(0)+g(0)+h(0)=1. C_1, π_1, π_2, A_1, B_1 are all constants, then Lemma III.1 follows.ư

Lemma III.2:

$\lim_{t \Downarrow \leftarrow} f_{A_2}(t) \mid C_2 \mid 1\,4\,\frac{\varsigma\psi}{\sigma_d\;2\;\sigma_s}m\,4\,\frac{\varsigma(1\,4\;\psi)}{\sigma_s}m$. *Furthermore, if* *$f_{A2}(0)=1$, g(0)=h(0)=0, then for any t>0, $f_{A2}(t)$∅C_2.*

Proof: From Figure 16, we can get the following equations:

$$\begin{cases} \frac{df(t)}{dt} \mid 4\varsigma m\,2\,(\sigma_d\;2\;\sigma_s)g(t)\,2\;\sigma_s h(t) \\ \frac{dg(t)}{dt} \mid \psi\varsigma m\,4\,(\sigma_d\;2\;\sigma_s)g(t) \\ h(t) \mid 1\,4\,f(t)\,4\,g(t) \end{cases}$$

Solve them, we have

$f_{A_2}(t) \mid A_2 e^{4\sigma_s t}\;2\;B_2 e^{4(\sigma_d 2 \sigma_s)t}\;2\;C_2$

where $\begin{cases} A_2 \mid \frac{(1\,4\;\psi)\varsigma}{\sigma_s}m\,4\,h(0) \\ B_2 \mid \frac{\psi\varsigma}{(\sigma_d\;2\;\sigma_s)}m\,4\,g(0) \\ C_2 \mid 1\,4\,\frac{(1\,4\;\psi)\varsigma}{\sigma_s}m\,4\,\frac{\psi\varsigma}{(\sigma_d\;2\;\sigma_s)}m \end{cases}$

If g(0)=h(0)=0, then $A_2>0$ and $B_2>0$; therefore, for any t>0, $f_{A2}(t)>C_2$. ư

Proof of Lemma 3.2.1:

Let $\varphi \mid \frac{\varsigma\psi}{(\sigma_d\;2\;\sigma_s)}\,2\,\frac{\varsigma(1\,4\;\psi)}{\sigma_s}$,

so $C_1 \mid \frac{1}{1\,2\,\varphi}$ and C_2=1-φm.

1) When 0ΩmΩC_1, with some algebra we can get C_2∅C_1. Because m<1 and f(0)=1, f(t) starts in world 2 and stays there as long as f(t)∅m. From Lemma III.2, &t>0 f(t)∅C_2∅C_1∅m, therefore it stays in A_2. The first part of Lemma 3.2.1 is proved.

2) When m>C_1, similarly we can get m>C_1>C_2. From Lemma III.1 and III.2 we know that)t^* such that &t>t^* f(t)=f_{A1}(t). Therefore $\lim_{t \Downarrow \leftarrow} f(t) \mid \lim_{t \Downarrow \leftarrow} f_{A_1}(t) \mid C_1$. The second part of Lemma 3.2.1 is proved.ư